

IEEE Std 1003.1™, 2003 Edition

The Open Group Technical Standard
Base Specifications, Issue 6

Includes IEEE Std 1003.1™-2001 and IEEE Std 1003.1™-2001/Cor 1-2002

Information Technology — Portable Operating System Interface (POSIX®)

The Base Specifications Issue 6

Incorporating Technical Corrigendum 1

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

and

The Open Group



THE *Open* GROUP

[This page intentionally left blank]

IEEE Std 1003.1™, 2003 Edition

The Open Group Technical Standard
Base Specifications, Issue 6

Includes IEEE Std 1003.1™-2001 and IEEE Std 1003.1™-2001/Cor 1-2002

Information Technology — Portable Operating System Interface (POSIX®)

Base Definitions

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

and

The Open Group



THE *Open* GROUP

[This page intentionally left blank]

Abstract

This standard is simultaneously ISO/IEC 9945:2002, IEEE Std 1003.1-2001, and forms the core of the Single UNIX Specification, Version 3.

The IEEE Std 1003.1, 2003 Edition includes IEEE Std 1003.1-2001/Cor 1-2002 incorporated into IEEE Std 1003.1-2001 (base document). The Corrigendum addresses problems discovered since the approval of IEEE Std 1003.1-2001. These changes are mainly due to resolving integration issues raised by the merger of the base documents that were incorporated into IEEE Std 1003.1-2001, which is the single common revision to IEEE Std 1003.1[™]-1996, IEEE Std 1003.2[™]-1992, ISO/IEC 9945-1:1996, ISO/IEC 9945-2:1993, and the Base Specifications of The Open Group Single UNIX[®] Specification, Version 2.

This standard defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. This standard is intended to be used by both applications developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of this standard:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

This standard describes the external characteristics and facilities that are of importance to applications developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, input/output (I/O), job control, network, portable operating system interface (POSIX[®]), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

Copyright © 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. All rights reserved. This printing is by the International Organization for Standardization with special permission of the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. Published in Switzerland.

Base Definitions, Issue 6

Published 31 March 2003 by the Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, U.S.A.
ISBN: 0-7381-3435-X PDF 0-7381-3564-X/SS95078 CD-ROM 0-7381-3563-1/SE95078
Printed in the United States of America by the IEEE.

Published 31 March 2003 by The Open Group
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, U.K.
Document Number: C031
ISBN: 1-931624-23-2
Printed in the U.K. by The Open Group.

All rights reserved. No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission from both the IEEE and The Open Group.

Portions of this standard are derived with permission from copyrighted material owned by Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc.

Permissions

Authorization to photocopy portions of this standard for internal or personal use is granted provided that the appropriate fee is paid to the Copyright Clearance Center or the equivalent body outside of the U.S. Permission to make multiple copies for educational purposes in the U.S. requires agreement and a license fee to be paid to the Copyright Clearance Center.

Beyond these provisions, permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests to the copyright holders should be sent by email to austin-group-permissions@opengroup.org.

Feedback

This standard has been prepared by the Austin Group. Feedback relating to the material contained in this standard may be submitted using the Austin Group web site at <http://www.opengroup.org/austin/defectform.html>.

IEEE

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property, or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “AS IS”.

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with the IEEE.¹ Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, U.S.A.

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE Standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

A patent holder has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and non-discriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Authorization to photocopy portions of any individual standard for internal or personal use is granted in the U.S. by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to the Copyright Clearance Center.² Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center. To arrange for payment of the licensing fee, please contact:

Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923, U.S.A., Tel.: +1 978 750 8400

Amendments, corrigenda, and interpretations for this standard, or information about the IEEE standards development process, may be found at <http://standards.ieee.org>.

Full catalog and ordering information on all IEEE publications is available from the IEEE Online Catalog & Store at <http://shop.ieee.org/store>.

1. For this standard, please send comments via the Austin Group as requested on page iii.

2. Please refer to the special provisions for this standard on page iii concerning permissions from both copyright holders and arrangements to cover photocopying and reproduction across the world, as well as by commercial organizations wishing to license the material for use in product documentation.

The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The Open Group's mission is to offer all organizations concerned with open information infrastructures a forum to share knowledge, integrate open initiatives, and certify approved products and processes in a manner in which they continue to trust our impartiality.

In the global eCommerce world of today, no single economic entity can achieve independence while still ensuring interoperability. The assurance that products will interoperate with each other across differing systems and platforms is essential to the success of eCommerce and business workflow. The Open Group, with its proven testing and certification program, is the international guarantor of interoperability in the new century.

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group's members include some of the largest and most influential organizations in the world. The flexible structure of The Open Groups membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information is available on The Open Group web site at <http://www.opengroup.org>.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes the *Westwood* family of tests for this standard and the associated certification program for Version 3 of the Single UNIX Specification, as well tests for CDE, CORBA, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at <http://www.opengroup.org/testing>.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at <http://www.opengroup.org/pubs>.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at <http://www.opengroup.org/corrigenda>.

Full catalog and ordering information on all Open Group publications is available at <http://www.opengroup.org/pubs>.

Contents

Chapter	1	Introduction.....	1
	1.1	Scope.....	1
	1.2	Conformance	4
	1.3	Normative References	4
	1.4	Terminology	5
	1.5	Portability	6
	1.5.1	Codes	6
	1.5.2	Margin Code Notation.....	14
 Chapter	 2	 Conformance.....	 17
	2.1	Implementation Conformance.....	17
	2.1.1	Requirements.....	17
	2.1.2	Documentation	17
	2.1.3	POSIX Conformance	18
	2.1.3.1	POSIX System Interfaces	18
	2.1.3.2	POSIX Shell and Utilities.....	20
	2.1.4	XSI Conformance	21
	2.1.4.1	XSI System Interfaces.....	21
	2.1.4.2	XSI Shell and Utilities Conformance	22
	2.1.5	Option Groups.....	22
	2.1.5.1	Subprofiling Considerations.....	22
	2.1.5.2	XSI Option Groups	24
	2.1.6	Options.....	28
	2.1.6.1	System Interfaces	29
	2.1.6.2	Shell and Utilities.....	29
	2.2	Application Conformance.....	31
	2.2.1	Strictly Conforming POSIX Application.....	31
	2.2.2	Conforming POSIX Application.....	32
	2.2.2.1	ISO/IEC Conforming POSIX Application.....	32
	2.2.2.2	<National Body> Conforming POSIX Application	32
	2.2.3	Conforming POSIX Application Using Extensions	32
	2.2.4	Strictly Conforming XSI Application	32
	2.2.5	Conforming XSI Application Using Extensions.....	33
	2.3	Language-Dependent Services for the C Programming Language..	33
	2.4	Other Language-Related Specifications.....	33
 Chapter	 3	 Definitions.....	 35
	3.1	Abortive Release	35
	3.2	Absolute Pathname	35
	3.3	Access Mode	35
	3.4	Additional File Access Control Mechanism.....	35
	3.5	Address Space	35

3.6	Advisory Information	35
3.7	Affirmative Response	36
3.8	Alert	36
3.9	Alert Character (<alert>)	36
3.10	Alias Name	36
3.11	Alignment	36
3.12	Alternate File Access Control Mechanism	36
3.13	Alternate Signal Stack	37
3.14	Ancillary Data	37
3.15	Angle Brackets	37
3.16	Application	37
3.17	Application Address	37
3.18	Application Program Interface (API)	37
3.19	Appropriate Privileges	37
3.20	Argument	38
3.21	Arm (a Timer)	38
3.22	Asterisk	38
3.23	Async-Cancel-Safe Function	38
3.24	Asynchronous Events	38
3.25	Asynchronous Input and Output	38
3.26	Async-Signal-Safe Function	38
3.27	Asynchronously-Generated Signal	39
3.28	Asynchronous I/O Completion	39
3.29	Asynchronous I/O Operation	39
3.30	Authentication	39
3.31	Authorization	39
3.32	Background Job	39
3.33	Background Process	39
3.34	Background Process Group (or Background Job)	39
3.35	Backquote	40
3.36	Backslash	40
3.37	Backspace Character (<backspace>)	40
3.38	Barrier	40
3.39	Base Character	40
3.40	Basename	40
3.41	Basic Regular Expression (BRE)	40
3.42	Batch Access List	40
3.43	Batch Administrator	41
3.44	Batch Client	41
3.45	Batch Destination	41
3.46	Batch Destination Identifier	41
3.47	Batch Directive	41
3.48	Batch Job	41
3.49	Batch Job Attribute	42
3.50	Batch Job Identifier	42
3.51	Batch Job Name	42
3.52	Batch Job Owner	42
3.53	Batch Job Priority	42

3.54	Batch Job State	42
3.55	Batch Name Service.....	42
3.56	Batch Name Space	42
3.57	Batch Node	43
3.58	Batch Operator	43
3.59	Batch Queue.....	43
3.60	Batch Queue Attribute.....	43
3.61	Batch Queue Position	43
3.62	Batch Queue Priority.....	43
3.63	Batch Rerunability	43
3.64	Batch Restart	44
3.65	Batch Server	44
3.66	Batch Server Name	44
3.67	Batch Service	44
3.68	Batch Service Request	44
3.69	Batch Submission.....	44
3.70	Batch System.....	44
3.71	Batch Target User.....	45
3.72	Batch User.....	45
3.73	Bind.....	45
3.74	Blank Character (<blank>).....	45
3.75	Blank Line	45
3.76	Blocked Process (or Thread).....	45
3.77	Blocking	45
3.78	Block-Mode Terminal.....	45
3.79	Block Special File.....	46
3.80	Braces.....	46
3.81	Brackets	46
3.82	Broadcast	46
3.83	Built-In Utility (or Built-In).....	46
3.84	Byte	46
3.85	Byte Input / Output Functions	47
3.86	Carriage-Return Character (<carriage-return>).....	47
3.87	Character	47
3.88	Character Array.....	47
3.89	Character Class.....	47
3.90	Character Set.....	47
3.91	Character Special File.....	48
3.92	Character String	48
3.93	Child Process	48
3.94	Circumflex.....	48
3.95	Clock.....	48
3.96	Clock Jump.....	48
3.97	Clock Tick.....	48
3.98	Coded Character Set.....	48
3.99	Codeset.....	49
3.100	Collating Element	49
3.101	Collation	49

3.102	Collation Sequence	49
3.103	Column Position	49
3.104	Command	50
3.105	Command Language Interpreter	50
3.106	Composite Graphic Symbol	50
3.107	Condition Variable	50
3.108	Connection	50
3.109	Connection Mode	50
3.110	Connectionless Mode	50
3.111	Control Character	51
3.112	Control Operator	51
3.113	Controlling Process	51
3.114	Controlling Terminal	51
3.115	Conversion Descriptor	51
3.116	Core File	51
3.117	CPU Time (Execution Time)	51
3.118	CPU-Time Clock	52
3.119	CPU-Time Timer	52
3.120	Current Job	52
3.121	Current Working Directory	52
3.122	Cursor Position	52
3.123	Datagram	52
3.124	Data Segment	52
3.125	Deferred Batch Service	52
3.126	Device	52
3.127	Device ID	52
3.128	Directory	53
3.129	Directory Entry (or Link)	53
3.130	Directory Stream	53
3.131	Disarm (a Timer)	53
3.132	Display	53
3.133	Display Line	53
3.134	Dollar Sign	53
3.135	Dot	53
3.136	Dot-Dot	54
3.137	Double-Quote	54
3.138	Downshifting	54
3.139	Driver	54
3.140	Effective Group ID	54
3.141	Effective User ID	54
3.142	Eight-Bit Transparency	54
3.143	Empty Directory	54
3.144	Empty Line	55
3.145	Empty String (or Null String)	55
3.146	Empty Wide-Character String	55
3.147	Encoding Rule	55
3.148	Entire Regular Expression	55
3.149	Epoch	55

3.150	Equivalence Class	55
3.151	Era	55
3.152	Event Management.....	56
3.153	Executable File.....	56
3.154	Execute	56
3.155	Execution Time.....	56
3.156	Execution Time Monitoring.....	56
3.157	Expand	56
3.158	Extended Regular Expression (ERE).....	56
3.159	Extended Security Controls	57
3.160	Feature Test Macro	57
3.161	Field	57
3.162	FIFO Special File (or FIFO)	57
3.163	File	57
3.164	File Description	58
3.165	File Descriptor	58
3.166	File Group Class.....	58
3.167	File Mode	58
3.168	File Mode Bits	58
3.169	Filename.....	58
3.170	Filename Portability	58
3.171	File Offset.....	59
3.172	File Other Class	59
3.173	File Owner Class	59
3.174	File Permission Bits.....	59
3.175	File Serial Number	59
3.176	File System	59
3.177	File Type.....	59
3.178	Filter	60
3.179	First Open (of a File).....	60
3.180	Flow Control	60
3.181	Foreground Job.....	60
3.182	Foreground Process	60
3.183	Foreground Process Group (or Foreground Job).....	60
3.184	Foreground Process Group ID	60
3.185	Form-Feed Character (<form-feed>)	60
3.186	Graphic Character.....	61
3.187	Group Database	61
3.188	Group ID.....	61
3.189	Group Name	61
3.190	Hard Limit.....	61
3.191	Hard Link	61
3.192	Home Directory.....	61
3.193	Host Byte Order	62
3.194	Incomplete Line.....	62
3.195	Inf	62
3.196	Instrumented Application.....	62
3.197	Interactive Shell.....	62

3.198	Internationalization.....	62
3.199	Interprocess Communication.....	62
3.200	Invoke.....	62
3.201	Job.....	63
3.202	Job Control	63
3.203	Job Control Job ID.....	63
3.204	Last Close (of a File)	63
3.205	Line	63
3.206	Linger.....	63
3.207	Link	63
3.208	Link Count.....	64
3.209	Local Customs	64
3.210	Local Interprocess Communication (Local IPC).....	64
3.211	Locale.....	64
3.212	Localization.....	64
3.213	Login.....	64
3.214	Login Name.....	64
3.215	Map	64
3.216	Marked Message	65
3.217	Matched	65
3.218	Memory Mapped Files.....	65
3.219	Memory Object.....	65
3.220	Memory-Resident	65
3.221	Message.....	65
3.222	Message Catalog	66
3.223	Message Catalog Descriptor.....	66
3.224	Message Queue	66
3.225	Mode.....	66
3.226	Monotonic Clock.....	66
3.227	Mount Point	66
3.228	Multi-Character Collating Element.....	66
3.229	Mutex.....	66
3.230	Name	67
3.231	Named STREAM.....	67
3.232	NaN (Not a Number).....	67
3.233	Native Language.....	67
3.234	Negative Response	67
3.235	Network	67
3.236	Network Address.....	67
3.237	Network Byte Order.....	68
3.238	Newline Character (<newline>)	68
3.239	Nice Value	68
3.240	Non-Blocking.....	68
3.241	Non-Spacing Characters	68
3.242	NUL	68
3.243	Null Byte.....	69
3.244	Null Pointer.....	69
3.245	Null String.....	69

3.246	Null Wide-Character Code	69
3.247	Number Sign	69
3.248	Object File	69
3.249	Octet.....	69
3.250	Offset Maximum	69
3.251	Opaque Address	69
3.252	Open File.....	70
3.253	Open File Description	70
3.254	Operand	70
3.255	Operator.....	70
3.256	Option	70
3.257	Option-Argument.....	70
3.258	Orientation	70
3.259	Orphaned Process Group.....	70
3.260	Page.....	71
3.261	Page Size	71
3.262	Parameter.....	71
3.263	Parent Directory	71
3.264	Parent Process.....	71
3.265	Parent Process ID	71
3.266	Pathname.....	72
3.267	Pathname Component.....	72
3.268	Path Prefix.....	72
3.269	Pattern	72
3.270	Period.....	72
3.271	Permissions	72
3.272	Persistence	72
3.273	Pipe	73
3.274	Polling.....	73
3.275	Portable Character Set.....	73
3.276	Portable Filename Character Set.....	73
3.277	Positional Parameter	73
3.278	Preallocation	73
3.279	Preempted Process (or Thread).....	74
3.280	Previous Job	74
3.281	Printable Character.....	74
3.282	Printable File	74
3.283	Priority.....	74
3.284	Priority Band	74
3.285	Priority Inversion	74
3.286	Priority Scheduling.....	74
3.287	Priority-Based Scheduling.....	75
3.288	Privilege	75
3.289	Process.....	75
3.290	Process Group.....	75
3.291	Process Group ID	75
3.292	Process Group Leader.....	75
3.293	Process Group Lifetime	75

3.294	Process ID	76
3.295	Process Lifetime	76
3.296	Process Memory Locking	76
3.297	Process Termination	76
3.298	Process-To-Process Communication	76
3.299	Process Virtual Time	76
3.300	Program.....	77
3.301	Protocol	77
3.302	Pseudo-Terminal	77
3.303	Radix Character.....	77
3.304	Read-Only File System	77
3.305	Read-Write Lock.....	77
3.306	Real Group ID.....	77
3.307	Real Time.....	78
3.308	Realtime Signal Extension.....	78
3.309	Real User ID	78
3.310	Record	78
3.311	Redirection	78
3.312	Redirection Operator	78
3.313	Reentrant Function	78
3.314	Referenced Shared Memory Object	78
3.315	Refresh.....	79
3.316	Regular Expression.....	79
3.317	Region.....	79
3.318	Regular File	79
3.319	Relative Pathname	79
3.320	Relocatable File.....	79
3.321	Relocation	79
3.322	Requested Batch Service.....	79
3.323	(Time) Resolution	79
3.324	Root Directory	80
3.325	Runnable Process (or Thread)	80
3.326	Running Process (or Thread).....	80
3.327	Saved Resource Limits.....	80
3.328	Saved Set-Group-ID	80
3.329	Saved Set-User-ID.....	80
3.330	Scheduling.....	80
3.331	Scheduling Allocation Domain.....	80
3.332	Scheduling Contention Scope	81
3.333	Scheduling Policy.....	81
3.334	Screen	81
3.335	Scroll	81
3.336	Semaphore.....	81
3.337	Session.....	81
3.338	Session Leader	82
3.339	Session Lifetime	82
3.340	Shared Memory Object	82
3.341	Shell.....	82

3.342	Shell, the.....	82
3.343	Shell Script.....	82
3.344	Signal	82
3.345	Signal Stack	83
3.346	Single-Quote	83
3.347	Slash.....	83
3.348	Socket	83
3.349	Socket Address.....	83
3.350	Soft Limit	83
3.351	Source Code	83
3.352	Space Character (<space>).....	84
3.353	Spawn.....	84
3.354	Special Built-In	84
3.355	Special Parameter.....	84
3.356	Spin Lock	84
3.357	Sporadic Server	84
3.358	Standard Error	84
3.359	Standard Input.....	84
3.360	Standard Output	84
3.361	Standard Utilities	85
3.362	Stream.....	85
3.363	STREAM	85
3.364	STREAM End	85
3.365	STREAM Head	85
3.366	STREAMS Multiplexor	85
3.367	String	85
3.368	Subshell.....	86
3.369	Successfully Transferred.....	86
3.370	Supplementary Group ID	86
3.371	Suspended Job	86
3.372	Symbolic Link.....	86
3.373	Synchronized Input and Output	86
3.374	Synchronized I/O Completion	86
3.375	Synchronized I/O Data Integrity Completion	87
3.376	Synchronized I/O File Integrity Completion	87
3.377	Synchronized I/O Operation	87
3.378	Synchronous I/O Operation	87
3.379	Synchronously-Generated Signal.....	87
3.380	System	87
3.381	System Crash	88
3.382	System Console	88
3.383	System Databases	88
3.384	System Documentation	88
3.385	System Process	88
3.386	System Reboot	89
3.387	System Trace Event.....	89
3.388	System-Wide.....	89
3.389	Tab Character (<tab>).....	89

3.390	Terminal (or Terminal Device)	89
3.391	Text Column.....	89
3.392	Text File	90
3.393	Thread	90
3.394	Thread ID.....	90
3.395	Thread List	90
3.396	Thread-Safe	90
3.397	Thread-Specific Data Key.....	90
3.398	Tilde	91
3.399	Timeouts	91
3.400	Timer	91
3.401	Timer Overrun.....	91
3.402	Token	91
3.403	Trace Analyzer Process	91
3.404	Trace Controller Process	91
3.405	Trace Event	91
3.406	Trace Event Type	91
3.407	Trace Event Type Mapping	92
3.408	Trace Filter	92
3.409	Trace Generation Version	92
3.410	Trace Log.....	92
3.411	Trace Point	92
3.412	Trace Stream.....	92
3.413	Trace Stream Identifier.....	92
3.414	Trace System	92
3.415	Traced Process	92
3.416	Tracing Status of a Trace Stream.....	93
3.417	Typed Memory Name Space	93
3.418	Typed Memory Object	93
3.419	Typed Memory Pool	93
3.420	Typed Memory Port	93
3.421	Unbind.....	93
3.422	Unit Data.....	93
3.423	Upshifting.....	93
3.424	User Database	94
3.425	User ID	94
3.426	User Name.....	94
3.427	User Trace Event	94
3.428	Utility.....	94
3.429	Variable	95
3.430	Vertical-Tab Character (<vertical-tab>)	95
3.431	White Space.....	95
3.432	Wide-Character Code (C Language)	95
3.433	Wide-Character Input / Output Functions.....	95
3.434	Wide-Character String	95
3.435	Word	96
3.436	Working Directory (or Current Working Directory)	96
3.437	Worldwide Portability Interface	96

3.438	Write	96
3.439	XSI	96
3.440	XSI-Conformant	96
3.441	Zombie Process	96
3.442	± 0	97
Chapter 4	General Concepts	99
4.1	Concurrent Execution	99
4.2	Directory Protection	99
4.3	Extended Security Controls	99
4.4	File Access Permissions	99
4.5	File Hierarchy	100
4.6	Filenames	100
4.7	File Times Update	100
4.8	Host and Network Byte Orders	101
4.9	Measurement of Execution Time	101
4.10	Memory Synchronization	102
4.11	Pathname Resolution	102
4.12	Process ID Reuse	103
4.13	Scheduling Policy	103
4.14	Seconds Since the Epoch	104
4.15	Semaphore	104
4.16	Thread-Safety	105
4.17	Tracing	105
4.18	Treatment of Error Conditions for Mathematical Functions	107
4.18.1	Domain Error	107
4.18.2	Pole Error	108
4.18.3	Range Error	108
4.18.3.1	Result Overflows	108
4.18.3.2	Result Underflows	108
4.19	Treatment of NaN Arguments for the Mathematical Functions	108
4.20	Utility	109
4.21	Variable Assignment	109
Chapter 5	File Format Notation	111
Chapter 6	Character Set	115
6.1	Portable Character Set	115
6.2	Character Encoding	118
6.3	C Language Wide-Character Codes	119
6.4	Character Set Description File	119
6.4.1	State-Dependent Character Encodings	122
Chapter 7	Locale	123
7.1	General	123
7.2	POSIX Locale	124
7.3	Locale Definition	124
7.3.1	LC_CTYPE	126

7.3.1.1	LC_CTYPE Category in the POSIX Locale	130
7.3.2	LC_COLLATE	134
7.3.2.1	The collating-element Keyword	135
7.3.2.2	The collating-symbol Keyword	136
7.3.2.3	The order_start Keyword	136
7.3.2.4	Collation Order	137
7.3.2.5	The order_end Keyword	139
7.3.2.6	LC_COLLATE Category in the POSIX Locale	139
7.3.3	LC_MONETARY	142
7.3.3.1	LC_MONETARY Category in the POSIX Locale	144
7.3.4	LC_NUMERIC	145
7.3.4.1	LC_NUMERIC Category in the POSIX Locale	146
7.3.5	LC_TIME	147
7.3.5.1	LC_TIME Locale Definition	147
7.3.5.2	LC_TIME C-Language Access	149
7.3.5.3	LC_TIME Category in the POSIX Locale	150
7.3.6	LC_MESSAGES	152
7.3.6.1	LC_MESSAGES Category in the POSIX Locale	153
7.4	Locale Definition Grammar	153
7.4.1	Locale Lexical Conventions	153
7.4.2	Locale Grammar	154
Chapter 8	Environment Variables	161
8.1	Environment Variable Definition	161
8.2	Internationalization Variables	162
8.3	Other Environment Variables	165
Chapter 9	Regular Expressions	169
9.1	Regular Expression Definitions	169
9.2	Regular Expression General Requirements	170
9.3	Basic Regular Expressions	171
9.3.1	BREs Matching a Single Character or Collating Element	171
9.3.2	BRE Ordinary Characters	171
9.3.3	BRE Special Characters	171
9.3.4	Periods in BREs	172
9.3.5	RE Bracket Expression	172
9.3.6	BREs Matching Multiple Characters	174
9.3.7	BRE Precedence	175
9.3.8	BRE Expression Anchoring	175
9.4	Extended Regular Expressions	175
9.4.1	EREs Matching a Single Character or Collating Element	176
9.4.2	ERE Ordinary Characters	176
9.4.3	ERE Special Characters	176
9.4.4	Periods in EREs	177
9.4.5	ERE Bracket Expression	177
9.4.6	EREs Matching Multiple Characters	177
9.4.7	ERE Alternation	178
9.4.8	ERE Precedence	178

	9.4.9	ERE Expression Anchoring.....	178
	9.5	Regular Expression Grammar.....	179
	9.5.1	BRE/ERE Grammar Lexical Conventions.....	179
	9.5.2	RE and Bracket Expression Grammar.....	180
	9.5.3	ERE Grammar.....	182
Chapter	10	Directory Structure and Devices	185
	10.1	Directory Structure and Files	185
	10.2	Output Devices and Terminal Types.....	185
Chapter	11	General Terminal Interface.....	187
	11.1	Interface Characteristics	187
	11.1.1	Opening a Terminal Device File	187
	11.1.2	Process Groups.....	187
	11.1.3	The Controlling Terminal.....	188
	11.1.4	Terminal Access Control	188
	11.1.5	Input Processing and Reading Data.....	189
	11.1.6	Canonical Mode Input Processing	189
	11.1.7	Non-Canonical Mode Input Processing.....	190
	11.1.8	Writing Data and Output Processing	191
	11.1.9	Special Characters.....	191
	11.1.10	Modem Disconnect.....	192
	11.1.11	Closing a Terminal Device File	192
	11.2	Parameters that Can be Set	193
	11.2.1	The termios Structure	193
	11.2.2	Input Modes.....	193
	11.2.3	Output Modes	194
	11.2.4	Control Modes.....	196
	11.2.5	Local Modes	197
	11.2.6	Special Control Characters	198
Chapter	12	Utility Conventions.....	201
	12.1	Utility Argument Syntax.....	201
	12.2	Utility Syntax Guidelines.....	203
Chapter	13	Headers.....	205
	13.1	Format of Entries.....	205
		Index.....	431
List of Tables			
	3-1	Job Control Job ID Formats.....	63
	5-1	Escape Sequences and Associated Actions	112
	6-1	Portable Character Set.....	115
	6-2	Control Character Set.....	120
	7-1	Valid Character Class Combinations.....	130
	10-1	Control Character Names	186

Foreword

Structure of the Standard

This standard was originally developed by the Austin Group, a joint working group of members of the IEEE, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1, as one of the four volumes of IEEE Std 1003.1-2001. The standard was approved by ISO and IEC and published in four parts, correlating to the original volumes.

A mapping of the parts to the volumes is shown below:

ISO/IEC 9945 Part	IEEE Std 1003.1 Volume	Description
9945-1	Base Definitions	Includes general terms, concepts, and interfaces common to all parts of ISO/IEC 9945, including utility conventions and C-language header definitions.
9945-2	System Interfaces	Includes definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery.
9945-3	Shell and Utilities	Includes definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs.
9945-4	Rationale	Includes extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of ISO/IEC 9945 and why features were included or discarded by the standard developers.

All four parts comprise the entire standard, and are intended to be used together to accommodate significant internal referencing among them. POSIX-conforming systems are required to support all four parts.

Introduction

Note: This introduction is not part of IEEE Std 1003.1-2001, Standard for Information Technology — Portable Operating System Interface (POSIX).

This standard has been jointly developed by the IEEE and The Open Group. It is simultaneously an IEEE Standard, an ISO/IEC Standard, and an Open Group Technical Standard.

The Austin Group

This standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.³ The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group has been to revise, combine, and update the following standards: ISO/IEC 9945-1, ISO/IEC 9945-2, IEEE Std 1003.1, IEEE Std 1003.2, and the Base Specifications of The Open Group Single UNIX Specification.

After two initial meetings, an agreement was signed in July 1999 between The Open Group and the Institute of Electrical and Electronics Engineers (IEEE), Inc., to formalize the project with the first draft of the revised specifications being made available at the same time. Under this agreement, The Open Group and IEEE agreed to share joint copyright of the resulting work. The Open Group has provided the chair and secretariat for the Austin Group.

The base document for the revision was The Open Group's Base volumes of its Single UNIX Specification, Version 2. These were selected since they were a superset of the existing POSIX.1 and POSIX.2 specifications and had some organizational aspects that would benefit the audience for the new revision.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group's Technical Standard designation, and an ISO/IEC designation. This set of specifications forms the core of the Single UNIX Specification, Version 3.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see <http://www.opengroup.org/austin>.

3. The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

Background

The developers of this standard represent a cross section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, this standard describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application writers to write portable applications—it was developed with that goal in mind—it has been designated POSIX,⁴ an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of standards.

Audience

The intended audience for this standard is all persons concerned with an industry-wide standard operating system based on the UNIX system. This includes at least four groups of people:

1. Persons buying hardware and software systems
2. Persons managing companies that are deciding on future corporate computing directions
3. Persons implementing operating systems, and especially
4. Persons developing applications where portability is an objective

Purpose

Several principles guided the development of this standard:

- Application-Oriented

The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. This standard codifies the common, existing definition of the UNIX system.

- Interface, Not Implementation

This standard defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.

4. The name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

- Source, Not Object, Portability

This standard has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. This standard does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.

- The C Language

The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.

- No Superuser, No System Administration

There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in this standard. This standard is also not concerned with hardware constraints or system maintenance.

- Minimal Interface, Minimally Defined

In keeping with the historical design principles of the UNIX system, the mandatory core facilities of this standard have been kept as minimal as possible. Additional capabilities have been added as optional extensions.

- Broadly Implementable

The developers of this standard endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:

1. All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
2. Compatible systems that are not derived from the original UNIX system code
3. Emulations hosted on entirely different operating systems
4. Networked systems
5. Distributed systems
6. Systems running on a broad range of hardware

No direct references to this goal appear in this standard, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations

When the original version of IEEE Std 1003.1 was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and the subsequent revisions and addenda to all of them have consolidated this consensus, and this revision reflects the significantly increased level of consensus arrived at since the original versions. The earlier standards and their modifications specified a number of areas where consensus had not been reached before, and these are now reflected in this revision. The authors of the original versions tried, as much as possible, to follow the principles below

when creating new specifications:

1. By standardizing an interface like one in an historical implementation; for example, directories
2. By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended *tar* format defined in the *pax* utility
3. By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the *sigaction()* function

This revision tries to minimize the number of changes required to implementations which conform to the earlier versions of the approved standards to bring them into conformance with the current standard. Specifically, the scope of this work excluded doing any “new” work, but rather collecting into a single document what had been spread across a number of documents, and presenting it in what had been proven in practice to be a more effective way. Some changes to prior conforming implementations were unavoidable, primarily as a consequence of resolving conflicts found in prior revisions, or which became apparent when bringing the various pieces together.

However, since it references the 1999 version of the ISO C standard, and no longer supports “Common Usage C”, there are a number of unavoidable changes. Applications portability is similarly affected.

This standard is specifically not a codification of a particular vendor’s product.

It should be noted that implementations will have different kinds of extensions. Some will reflect “historical usage” and will be preserved for execution of pre-existing applications. These functions should be considered “obsolescent” and the standard functions used for new applications. Some extensions will represent functions beyond the scope of this standard. These need to be used with careful management to be able to adapt to future extensions of this standard and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code

A goal of this standard was to minimize additional work for the developers of applications. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

This Standard

This standard defines the Portable Operating System Interface (POSIX) requirements and consists of the following volumes:

- Base Definitions (this volume)
- Shell and Utilities
- System Interfaces
- Rationale (Informative)

This Volume

The Base Definitions volume provides common definitions for this standard, therefore readers should be familiar with it before using the other volumes.

This volume is structured as follows:

- Chapter 1 is an introduction.
- Chapter 2 defines the conformance requirements.
- Chapter 3 defines general terms used.
- Chapter 4 describes general concepts used.
- Chapter 5 describes the notation used to specify file input and output formats in this volume and the Shell and Utilities volume.
- Chapter 6 describes the portable character set and the process of character set definition.
- Chapter 7 describes the syntax for defining internationalization locales as well as the POSIX locale provided on all systems.
- Chapter 8 describes the use of environment variables for internationalization and other purposes.
- Chapter 9 describes the syntax of pattern matching using regular expressions employed by many utilities and matched by the *regcomp()* and *regex()* functions.
- Chapter 10 describes files and devices found on all systems.
- Chapter 11 describes the asynchronous terminal interface for many of the functions in the System Interfaces volume and the *stty* utility in the Shell and Utilities volume.
- Chapter 12 describes the policies for command line argument construction and parsing.
- Chapter 13 defines the contents of headers which declare constants, macros, and data structures that are needed by programs using the services provided by the System Interfaces volume.

Comprehensive references are available in the index.

Typographical Conventions

The following typographical conventions are used throughout this standard. In the text, this standard is referred to as IEEE Std 1003.1-2001, which is technically identical to The Open Group Base Specifications, Issue 6.

The typographical conventions listed here are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this standard.

Reference	Example	Notes
C-Language Data Structure	aio	
C-Language Data Structure Member	<i>aio_lio_opcode</i>	
C-Language Data Type	long	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	

Reference	Example	Notes
C-Language Function Argument	<i>arg1</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	<sys/stat.h>	
C-Language Keyword	return	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	INET_ADDRSTRLEN	
C-Language Preprocessing Directive	#define	
Commands within a Utility	a, c	
Conversion Specification, Specifier / Modifier Character	%A, g, E	1
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	Hello, World	
Filename	/tmp	
Literal Character	'c', '\r', '\'	2
Literal String	"abcde"	2
Optional Items in Utility Syntax	[]	
Parameter	<directory pathname>	
Special Character	<newline>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	#include <sys/stat.h>	
User Input and Example Code	echo Hello, World	5
Utility Name	<i>awk</i>	
Utility Operand	<i>file_name</i>	
Utility Option	-c	
Utility Option with Option-Argument	-w width	

Notes:

1. Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf* and *fscanf* formatting functions.
2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. For literal characters, `'\'` (or any of the other sequences such as `''`) is the same as the C constant `'\\'` (or `'\'`).
3. The style selected for some of the special characters, such as `<newline>`, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters `<tab>` or `<newline>`.
4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
5. Brackets shown in this font, `" [] "`, are part of the syntax and do *not* indicate optional items. In syntax the `' | '` symbol is used to separate alternatives, and ellipses (`" . . . "`) are used to show that additional arguments are optional.

Shading is used to identify extensions and options; see Section 1.5.1 (on page 6).

Footnotes and notes within the body of the normative text are for information only (informative).

Informative sections (such as Rationale, Change History, Application Usage, and so on) are denoted by continuous shading bars in the margins.

Ranges of values are indicated with parentheses or brackets as follows:

- (a,b) means the range of all values from a to b , including neither a nor b .
- $[a,b]$ means the range of all values from a to b , including a and b .
- $[a,b)$ means the range of all values from a to b , including a , but not b .
- $(a,b]$ means the range of all values from a to b , including b , but not a .

Note: A symbolic limit beginning with POSIX is treated differently, depending on context. In a C-language header, the symbol `POSIXstring` (where *string* may contain underscores) is represented by the C identifier `_POSIXstring`, with a leading underscore required to prevent ISO C standard name space pollution. However, in other contexts, such as languages other than C, the leading underscore is not used because this requirement does not exist.

Participants

IEEE Std 1003.1-2001 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/SC22 WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair

Donald W. Cragun, Organizational Representative, IEEE PASC

Nicholas Stoughton, Organizational Representative, ISO/SC22 WG15

Mark Brown, Organizational Representative, The Open Group

Cathy Hughes, Technical Editor

Austin Group Technical Reviewers

Peter Anvin	Michael Gonzalez	Sandra O'Donnell
Bouazza Bachar	Joseph M. Gwinn	Frank Prindle
Theodore P. Baker	Jon Hitchcock	Curtis Royster Jr.
Walter Briscoe	Yvette Ho Sang	Glen Seeds
Mark Brown	Cathy Hughes	Keld Jorn Simonsen
Dave Butenhof	Lowell G. Johnson	Raja Srinivasan
Geoff Clare	Andrew Josey	Nicholas Stoughton
Donald W. Cragun	Michael Kavanaugh	Donn S. Terry
Lee Damico	David Korn	Fred Tydeman
Ulrich Drepper	Marc Aurele La France	Peter Van Der Veen
Paul Eggert	Jim Meyering	James Youngman
Joanna Farley	Gary Miller	Jim Zepeda
Clive D.W. Feather	Finnbarr P. Murphy	Jason Zions
Andrew Gollan	Joseph S. Myers	

Austin Group Working Group Members

Harold C. Adams	Michael Gonzalez	Sandra O'Donnell
Peter Anvin	Karen D. Gordon	Frank Prindle
Pierre-Jean Arcos	Joseph M. Gwinn	Francois Riche
Jay Ashford	Steven A. Haaser	John D. Riley
Bouazza Bachar	Charles E. Hammons	Andrew K. Roach
Theodore P. Baker	Chris J. Harding	Helmut Roth
Robert Barned	Barry Hedquist	Jaideep Roy
Joel Berman	Vincent E. Henley	Curtis Royster Jr.
David J. Blackwood	Karl Heubaum	Stephen C. Schwarm
Shirley Bockstahler-Brandt	Jon Hitchcock	Glen Seeds
James Bottomley	Yvette Ho Sang	Richard Seibel
Walter Briscoe	Niklas Holsti	David L. Shroads Jr.
Andries Brouwer	Thomas Hosmer	W. Olin Sibert
Mark Brown	Cathy Hughes	Keld Jorn Simonsen
Eric W. Burger	Jim D. Isaak	Curtis Smith
Alan Burns	Lowell G. Johnson	Raja Srinivasan
Andries Brouwer	Michael B. Jones	Nicholas Stoughton
Dave Butenhof	Andrew Josey	Marc J. Teller
Keith Chow	Michael J. Karels	Donn S. Terry
Geoff Clare	Michael Kavanaugh	Fred Tydeman
Donald W. Cragun	David Korn	Mark-Rene Uchida
Lee Damico	Steven Kramer	Scott A. Valcourt
Juan Antonio De La Puente	Thomas M. Kurihara	Peter Van Der Veen
Ming De Zhou	Marc Aurele La France	Michael W. Vannier
Steven J. Dovich	C. Douglass Locke	Eric Vought
Richard P. Draves	Nick Maclaren	Frederick N. Webb
Ulrich Drepper	Roger J. Martin	Paul A.T. Wolfgang
Paul Eggert	Craig H. Meyer	Garrett A. Wollman
Philip H. Enslow	Jim Meyering	James Youngman
Joanna Farley	Gary Miller	Oren Yuen
Clive D.W. Feather	Finnbarr P. Murphy	Janusz Zalewski
Pete Forman	Joseph S. Myers	Jim Zepeda
Mark Funkenhauser	John Napier	Jason Zions
Lois Goldthwaite	Peter E. Obermayer	
Andrew Gollan	James T. Oblinger	

The Open Group

When The Open Group approved the Base Specifications, Issue 6 on 12 September 2001, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair

Finnbarr P. Murphy, Vice-Chair

Mark Brown, Austin Group Liaison

Cathy Hughes, Technical Editor

Base Working Group Members

Bouazza Bachar

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Joanna Farley

Andrew Gollan

Karen D. Gordon

Gary Miller

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair
Joseph M. Gwinn, Vice-Chair
Jay Ashford, Functional Chair
Andrew Josey, Functional Chair
Curtis Royster Jr., Functional Chair
Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001. Balloters may have voted for approval, disapproval, or abstention:

Harold C. Adams	Steven A. Haaser	Frank Prindle
Pierre-Jean Arcos	Charles E. Hammons	Francois Riche
Jay Ashford	Chris J. Harding	John D. Riley
Theodore P. Baker	Barry Hedquist	Andrew K. Roach
Robert Barned	Vincent E. Henley	Helmut Roth
David J. Blackwood	Karl Heubaum	Jaideep Roy
Shirley Bockstahler-Brandt	Niklas Holsti	Curtis Royster Jr.
James Bottomley	Thomas Hosmer	Stephen C. Schwarm
Mark Brown	Jim D. Isaak	Richard Seibel
Eric W. Burger	Lowell G. Johnson	David L. Shroads Jr.
Alan Burns	Michael B. Jones	W. Olin Sibert
Dave Butenhof	Andrew Josey	Keld Jorn Simonsen
Keith Chow	Michael J. Karels	Nicholas Stoughton
Donald W. Cragun	Steven Kramer	Donn S. Terry
Juan Antonio De La Puente	Thomas M. Kurihara	Mark-Rene Uchida
Ming De Zhou	C. Douglass Locke	Scott A. Valcourt
Steven J. Dovich	Roger J. Martin	Michael W. Vannier
Richard P. Draves	Craig H. Meyer	Frederick N. Webb
Philip H. Enslow	Finnbarr P. Murphy	Paul A.T. Wolfgang
Michael Gonzalez	John Napier	Oren Yuen
Karen D. Gordon	Peter E. Obermayer	Janusz Zalewski
Joseph M. Gwinn	James T. Oblinger	

The following organizational representative voted on this standard:

Andrew Josey, X/Open Company Ltd.

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, it had the following membership:

Donald N. Heirman, Chair

James T. Carlo, Vice-Chair

Judith Gorman, Secretary

Satish K. Aggarwal

Mark D. Bowman

Gary R. Engmann

Harold E. Epstein

H. Landis Floyd

Jay Forster*

Howard M. Frazier

Ruben D. Garzon

James H. Gurney

Richard J. Holleman

Lowell G. Johnson

Robert J. Kennelly

Joseph L. Koepfinger*

Peter H. Lips

L. Bruce McClung

Daleep C. Mohla

James W. Moore

Robert F. Munzner

Ronald C. Petersen

Gerald H. Peterson

John B. Posey

Gary S. Robinson

Akio Tojo

Donald W. Zipse

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative

Donald R. Volzka, TAB Representative

Yvette Ho Sang, **Don Messina**, **Savoula Amanatidis**, IEEE Project Editors

* Member Emeritus

IEEE Std 1003.1-2001/Cor 1-2002 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/IEC JTC 1/SC22/WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair

Donald W. Cragun, Organizational Representative, IEEE PASC

Nicholas Stoughton, Organizational Representative, ISO/IEC JTC 1/SC22/WG15

Mark Brown, Organizational Representative, The Open Group

Cathy Fox, Technical Editor

Austin Group Technical Reviewers

Theodore P. Baker

Julian Blake

Andries Brouwer

Mark Brown

Dave Butenhof

Geoff Clare

Donald W. Cragun

Ken Dawson

Ulrich Drepper

Larry Dwyer

Paul Eggert

Joanna Farley

Clive D.W. Feather

Cathy Fox

Mark Funkenhauser

Lois Goldthwaite

Andrew Gollan

Michael Gonzalez

Bruno Haible

Ben Harris

Jon Hitchcock

Andreas Jaeger

Andrew Josey

Jonathan Lennox

Nick Maclaren

Jack McCann

Wilhelm Mueller

Joseph S. Myers

Frank Prindle

Kenneth Raeburn

Tim Robbins

Glen Seeds

Matthew Seitz

Keld Jorn Simonsen

Nicholas Stoughton

Alexander Terekhov

Donn S. Terry

Mike Wilson

Garrett A. Wollman

Mark Ziegast

Austin Group Working Group Members

Harold C. Adams
Alejandro Alonso
Jay Ashford
Theodore P. Baker
David J. Blackwood
Julian Blake
Mitchell Bonnett
Andries Brouwer
Mark Brown
Eric W. Burger
Alan Burns
Dave Butenhof
Keith Chow
Geoff Clare
Luis Cordova
Donald W. Cragun
Dragan Cvetkovic
Lee Damico
Ken Dawson
Jeroen Dekkers
Juan Antonio De La Puente
Steven J. Dovich
Ulrich Drepper
Dr. Sourav Dutta
Larry Dwyer
Paul Eggert
Joanna Farley

Clive D.W. Feather
Yaacov Fenster
Cathy Fox
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan
Michael Gonzalez
Karen D. Gordon
Scott Gudgel
Joseph M. Gwinn
Steven A. Haaser
Bruno Haible
Charles E. Hammons
Bryan Harold
Ben Harris
Barry Hedquist
Karl Heubaum
Jon Hitchcock
Andreas Jaeger
Andrew Josey
Kenneth Lang
Pi-Cheng Law
Jonathan Lennox
Nick Maclaren
Roger J. Martin
Jack McCann
George Miao

Wilhelm Mueller
Finnbarr P. Murphy
Joseph S. Myers
Alexey Neyman
Charles Ngethe
Peter Petrov
Frank Prindle
Vikram Punj
Kenneth Raeburn
Francois Riche
Tim Robbins
Curtis Royster Jr.
Diane Schleicher
Gil Shultz
Stephen C. Schwarm
Glen Seeds
Matthew Seitz
Keld Jorn Simonsen
Doug Stevenson
Nicholas Stoughton
Alexander Terekhov
Donn S. Terry
Mike Wilson
Garrett A. Wollman
Oren Yuen
Mark Ziegast

The Open Group

When The Open Group approved the Base Specifications, Issue 6, Technical Corrigendum 1 on 7 February 2003, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair

Finnbarr P. Murphy, Vice-Chair

Mark Brown, Austin Group Liaison

Cathy Fox, Technical Editor

Base Working Group Members

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Ulrich Drepper

Joanna Farley

Andrew Gollan

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001/Cor 1-2002 on 11 December 2002, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair

Joseph M. Gwinn, Vice-Chair

Jay Ashford, Functional Chair

Andrew Josey, Functional Chair

Curtis Royster Jr., Functional Chair

Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001/Cor 1-2002. Balloters may have voted for approval, disapproval, or abstention:

Alejandro Alonso

Jay Ashford

David J. Blackwood

Julian Blake

Mitchell Bonnett

Mark Brown

Dave Butenhof

Keith Chow

Luis Cordova

Donald W. Cragun

Steven J. Dovich

Dr. Sourav Dutta

Yaacov Fenster

Michael Gonzalez

Scott Gudgel

Charles E. Hammons

Bryan Harold

Barry Hedquist

Karl Heubaum

Lowell G. Johnson

Andrew Josey

Kenneth Lang

Pi-Cheng Law

George Miao

Roger J. Martin

Finnbarr P. Murphy

Charles Ngethe

Peter Petrov

Frank Prindle

Vikram Punj

Francois Riche

Curtis Royster Jr.

Diane Schleicher

Stephen C. Schwarm

Gil Shultz

Nicholas Stoughton

Donn S. Terry

Oren Yuen

Juan A. de la Puente

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 / Cor 1-2002 on 11 December 2002, the membership was as follows:

James T. Carlo, Chair

James H. Gurney, Vice-Chair

Judith Gorman, Secretary

Sid Bennett

H. Stephen Berger

Clyde R. Camp

Richard DeBlasio

Harold E. Epstein

Julian Forster*

Howard M. Frazier

Toshio Fukuda

Arnold M. Greenspan

Raymond Hapeman

Donald M. Heirman

Richard H. Hulett

Lowell G. Johnson

Joseph L. Koepfinger*

Peter H. Lips

Nader Mehravari

Daleep C. Mohla

William J. Moylan

Malcolm V. Thaden

Geoffrey O. Thompson

Howard L. Wolfman

Don Wright

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative

Satish K. Aggarwal, NRC Representative

Savoula Amanatidis, IEEE Standards Managing Editor

* Member Emeritus

Trademarks

The following information is given for the convenience of users of this standard and does not constitute endorsement of these products by The Open Group or the IEEE. There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

1003.1[™] is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

AIX[®] is a registered trademark of IBM Corporation.

AT&T[®] is a registered trademark of AT&T in the U.S.A. and other countries.

BSD[™] is a trademark of the University of California, Berkeley, U.S.A.

Hewlett-Packard[®], HP[®], and HP-UX[®] are registered trademarks of Hewlett-Packard Company.

IBM[®] is a registered trademark of International Business Machines Corporation.

The Open Group and Boundaryless Information Flow are trademarks and UNIX is a registered trademark of The Open Group in the United States and other countries. All other trademarks are the property of their respective owners.

POSIX[®] is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Sun[®] and Sun Microsystems[®] are registered trademarks of Sun Microsystems, Inc.

/usr/group[®] is a registered trademark of UniForum, the International Network of UNIX System Users.

Acknowledgements

The contributions of the following organizations to the development of IEEE Std 1003.1-2001 are gratefully acknowledged:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.
- The SC22 WG14 Committees.

This standard was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO SC22 WG15.

Referenced Documents

Normative References

Normative references for this standard are defined in Section 1.3 (on page 4).

Informative References

The following documents are referenced in this standard:

1984 /usr/group Standard

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

Almasi and Gottlieb

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

ANSI C

American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.226-1994

American National Standard for Information Systems: Standard X3.226-1994, Programming Language Common LISP.

Brawer

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989, ISBN: 0-12-128470-0.

DeRemer and Pennello Article

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets*, SigPlan Notices, Volume 15, No. 8, August 1979.

Draft ANSI X3J11.1

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-1

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

FIPS 151-2

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

HP-UX Manual

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

IEC 60559: 1989

IEC 60559: 1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously designated IEC 559: 1989).

IEEE Std 754-1985

IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic.

IEEE Std 854-1987

IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.

- IEEE Std 1003.9-1992
IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.
- IETF RFC 791
Internet Protocol, Version 4 (IPv4), September 1981.
- IETF RFC 819
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982.
- IETF RFC 822
Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982.
- IETF RFC 919
Broadcasting Internet Datagrams, J. Mogul, October 1984.
- IETF RFC 920
Domain Requirements, J. Postel, J. Reynolds, October 1984.
- IETF RFC 921
Domain Name System Implementation Schedule, J. Postel, October 1984.
- IETF RFC 922
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984.
- IETF RFC 1034
Domain Names — Concepts and Facilities, P. Mockapetris, November 1987.
- IETF RFC 1035
Domain Names — Implementation and Specification, P. Mockapetris, November 1987.
- IETF RFC 1123
Requirements for Internet Hosts — Application and Support, R. Braden, October 1989.
- IETF RFC 1886
DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson, December 1995.
- IETF RFC 2045
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996.
- IETF RFC 2181
Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997.
- IETF RFC 2373
Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998.
- IETF RFC 2460
Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998.
- Internationalisation Guide
Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.
- ISO C (1990)
ISO/IEC 9899:1990, Programming Languages — C, including Amendment 1:1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).

ISO 2375:1985

ISO 2375:1985, Data Processing — Procedure for Registration of Escape Sequences.

ISO 8652:1987

ISO 8652:1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).

ISO/IEC 1539:1990

ISO/IEC 1539:1990, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

ISO/IEC 4873:1991

ISO/IEC 4873:1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.

ISO/IEC 6429:1992

ISO/IEC 6429:1992, Information Technology — Control Functions for Coded Character Sets.

ISO/IEC 6937:1994

ISO/IEC 6937:1994, Information Technology — Coded Character Set for Text Communication — Latin Alphabet.

ISO/IEC 8802-3:1996

ISO/IEC 8802-3:1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

ISO/IEC 8859

ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

Part 1: Latin Alphabet No. 1

Part 2: Latin Alphabet No. 2

Part 3: Latin Alphabet No. 3

Part 4: Latin Alphabet No. 4

Part 5: Latin/Cyrillic Alphabet

Part 6: Latin/Arabic Alphabet

Part 7: Latin/Greek Alphabet

Part 8: Latin/Hebrew Alphabet

Part 9: Latin Alphabet No. 5

Part 10: Latin Alphabet No. 6

Part 13: Latin Alphabet No. 7

Part 14: Latin Alphabet No. 8

Part 15: Latin Alphabet No. 9

ISO POSIX-1:1996

ISO/IEC 9945-1:1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Std 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.

ISO POSIX-2:1993

ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2-1992, as amended by ANSI/IEEE Std 1003.2a-1992).

Issue 1

X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

Issue 2

X/Open Portability Guide, January 1987:

- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
- Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)

Issue 3

X/Open Specification, 1988, 1989, February 1992:

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

Issue 4

CAE Specification, July 1992, published by The Open Group:

- System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
- Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
- System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

Issue 4, Version 2

CAE Specification, August 1994, published by The Open Group:

- System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

Issue 5

Technical Standard, February 1997, published by The Open Group:

- System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

Knuth Article

Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control, Volume 8, No. 6, October 1965.

KornShell

Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming Language*, March 1995, Prentice Hall.

MSE Working Draft

Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

POSIX.0: 1995

IEEE Std 1003.0-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical to ISO/IEC TR 14252).

POSIX.1: 1988

IEEE Std 1003.1-1988, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1: 1990

IEEE Std 1003.1-1990, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1a

P1003.1a, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — (C Language) Amendment.

POSIX.1d: 1999

IEEE Std 1003.1d-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 4: Additional Realtime Extensions [C Language].

POSIX.1g: 2000

IEEE Std 1003.1g-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 6: Protocol-Independent Interfaces (PII).

POSIX.1j: 2000

IEEE Std 1003.1j-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 5: Advanced Realtime Extensions [C Language].

POSIX.1q: 2000

IEEE Std 1003.1q-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 7: Tracing [C Language].

POSIX.2b

P1003.2b, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment.

POSIX.2d:-1994

IEEE Std 1003.2d-1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.

POSIX.13:-1998

IEEE Std 1003.13:1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.

Sarwate Article

Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications of the ACM, Volume 30, No. 8, August 1988.

Sprunt, Sha, and Lehoczky

Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*, The Journal of Real-Time Systems, Volume 1, 1989, Pages 27-60.

SVID, Issue 1

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.

SVID, Issue 2

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.

SVID, Issue 3

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.

The AWK Programming Language

Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.

UNIX Programmer's Manual

American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.

XNS, Issue 4

CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.

XNS, Issue 5

CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XNS, Issue 5.2

Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.

X/Open Curses, Issue 4, Version 2

CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.

Yacc

Yacc: Yet Another Compiler Compiler, Stephen C. Johnson, 1978.

Source Documents

Parts of the following documents were used to create the base documents for this standard:

AIX 3.2 Manual

AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).

OSF/1

OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).

OSF AES

Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

System V Release 2.0

— UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).

— UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).

System V Release 4.2

Operating System API Reference, UNIX SVR4.2 (1992) (ISBN: 0-13-017658-3).

Introduction

1.1 Scope

IEEE Std 1003.1-2001 defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. It is intended to be used by both applications developers and system implementors.

IEEE Std 1003.1-2001 comprises four major components (each in an associated volume):

1. General terms, concepts, and interfaces common to all volumes of IEEE Std 1003.1-2001, including utility conventions and C-language header definitions, are included in the Base Definitions volume of IEEE Std 1003.1-2001.
2. Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume of IEEE Std 1003.1-2001.
3. Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume of IEEE Std 1003.1-2001.
4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of IEEE Std 1003.1-2001 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of IEEE Std 1003.1-2001.

The following areas are outside of the scope of IEEE Std 1003.1-2001:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

IEEE Std 1003.1-2001 describes the external characteristics and facilities that are of importance to applications developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

The facilities provided in IEEE Std 1003.1-2001 are drawn from the following base documents:

- IEEE Std 1003.1-1996 (POSIX-1) (incorporating IEEE Std 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995)
- The following amendments to the POSIX.1-1990 standard:
 - IEEE P1003.1a draft standard (Additional System Services)
 - IEEE Std 1003.1d-1999 (Additional Realtime Extensions)

- IEEE Std 1003.1g-2000 (Protocol-Independent Interfaces (PII))
- IEEE Std 1003.1j-2000 (Advanced Realtime Extensions)
- IEEE Std 1003.1q-2000 (Tracing)
- IEEE Std 1003.2-1992 (POSIX-2) (includes IEEE Std 1003.2a-1992)
- The following amendments to the ISO POSIX-2: 1993 standard:
 - IEEE P1003.2b draft standard (Additional Utilities)
 - IEEE Std 1003.2d-1994 (Batch Environment)
- Open Group Technical Standard, February 1997, System Interface Definitions, Issue 5 (XBD5) (ISBN: 1-85912-186-1, C605)
- Open Group Technical Standard, February 1997, Commands and Utilities, Issue 5 (XCU5) (ISBN: 1-85912-191-8, C604)
- Open Group Technical Standard, February 1997, System Interfaces and Headers, Issue 5 (XSH5) (in 2 Volumes) (ISBN: 1-85912-181-0, C606)
- Note:** XBD5, XCU5, and XSH5 are collectively referred to as the *Base Specifications*.
- Open Group Technical Standard, January 2000, Networking Services, Issue 5.2 (XNS5.2) (ISBN: 1-85912-241-8, C808)
- ISO / IEC 9899: 1999, Programming Languages — C.

IEEE Std 1003.1-2001 uses the Base Specifications as its organizational basis and adds the following additional functionality to them, drawn from the base documents above:

- Normative text from the ISO POSIX-1: 1996 standard and the ISO POSIX-2: 1993 standard not included in the Base Specifications
- The amendments to the POSIX.1-1990 standard and the ISO POSIX-2: 1993 standard listed above, except for parts of IEEE Std 1003.1g-2000
- Portability Considerations
- Additional rationale and notes

The following features, marked legacy or obsolescent in the base documents, are not carried forward into IEEE Std 1003.1-2001. Other features from the base documents marked legacy or obsolescent are carried forward unless otherwise noted.

From XSH5, the following legacy interfaces, headers, and external variables are not carried forward:

```
advance(), brk(), chroot(), compile(), cuserid(), gamma(), getdtablesize(), getpagesize(), getpass(),
getw(), putw(), re_comp(), re_exec(), regcmp(), regex(), sbrk(), sigstack(), step(), ttyslot(),
valloc(), wait3(), <re_comp.h>, <regexp.h>, <varargs.h>, loc1, __loc1, loc2, locs
```

From XCU5, the following legacy utilities are not carried forward:

```
calendar, cancel, cc, col, cpio, cu, dircmp, dis, egrep, fgrep, line, lint, lpstat, mail, pack, pcat, pg, spell,
sum, tar, unpack, uulog, uuname, uupick, uuto
```

In addition, legacy features within non-legacy reference pages (for example, headers) are not carried forward.

From the ISO POSIX-1: 1996 standard, the following obsolescent features are not carried forward:

78	Page 112, CLK_TCK
79	Page 197 <i>tcgetattr()</i> rate returned option
80	From the ISO POSIX-2: 1993 standard, obsolescent features within the following pages are not
81	carried forward:
82	Page 75, zero-length prefix within <i>PATH</i>
83	Page 156, 159 <i>set</i>
84	Page 178, <i>awk</i> , use of no argument and no parentheses with length
85	Page 259, <i>ed</i>
86	Page 272, <i>env</i>
87	Page 282, <i>find</i> perm [<i>-</i>]onum
88	Page 295-296, <i>egrep</i>
89	Page 299-300, <i>head</i>
90	Page 305-306, <i>join</i>
91	Page 309-310, <i>kill</i>
92	Page 431-433, 435-436, <i>sort</i>
93	Page 444-445, <i>tail</i>
94	Page 453, 455-456, <i>touch</i>
95	Page 464-465, <i>tty</i>
96	Page 472, <i>uniq</i>
97	Page 515-516, <i>ex</i>
98	Page 542-543, <i>expand</i>
99	Page 563-565, <i>more</i>
100	Page 574-576, <i>newgrp</i>
101	Page 578, <i>nice</i>
102	Page 594-596, <i>renice</i>
103	Page 597-598, <i>split</i>
104	Page 600-601, <i>strings</i>
105	Page 624-625, <i>vi</i>
106	Page 693, <i>lex</i>
107	The <i>c89</i> utility (which specified a compiler for the C Language specified by the
108	ISO/IEC 9899: 1990 standard) has been replaced by a <i>c99</i> utility (which specifies a compiler for
109	the C Language specified by the ISO/IEC 9899: 1999 standard).
110	From XSH5, text marked OH (Optional Header) has been reviewed on a case-by-case basis and
111	removed where appropriate. The XCU5 text marked OF (Output Format Incompletely Specified)
112	and UN (Possibly Unsupportable Feature) has been reviewed on a case-by-case basis and
113	removed where appropriate.
114	For the networking interfaces, the base document is the XNS, Issue 5.2 specification. The
115	following parts of the XNS, Issue 5.2 specification are out of scope and not included in
116	IEEE Std 1003.1-2001:
117	• Part 3 (XTI)
118	• Part 4 (Appendixes)
119	Since there is much duplication between the XNS, Issue 5.2 specification and
120	IEEE Std 1003.1g-2000, material only from the following sections of IEEE Std 1003.1g-2000 has
121	been included:
122	• General terms related to sockets (Section 2.2.2)
123	• Socket concepts (Sections 5.1 through 5.3, inclusive)

- The *pselect()* function (Sections 6.2.2.1 and 6.2.3)
- The *socketmark()* function (Section 5.4.13)
- The **<sys/select.h>** header (Section 6.2)

Emphasis is placed on standardizing existing practice for existing users, with changes and additions limited to correcting deficiencies in the following areas:

- Issues raised by IEEE or ISO / IEC Interpretations against IEEE Std 1003.1 and IEEE Std 1003.2
- Issues raised in corrigenda for the Base Specifications and working group resolutions from The Open Group
- Corrigenda and resolutions passed by The Open Group for the XNS, Issue 5.2 specification
- Changes to make the text self-consistent with the additional material merged
- A reorganization of the options in order to facilitate profiling, both for smaller profiles such as IEEE Std 1003.13, and larger profiles such as the Single UNIX Specification
- Alignment with the ISO / IEC 9899: 1999 standard

1.2 Conformance

Conformance requirements for IEEE Std 1003.1-2001 are defined in Chapter 2 (on page 17).

1.3 Normative References

The following standards contain provisions which, through references in IEEE Std 1003.1-2001, constitute provisions of IEEE Std 1003.1-2001. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on IEEE Std 1003.1-2001 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ANS X3.9-1978

(Reaffirmed 1989) American National Standard for Information Systems: Standard X3.9-1978, Programming Language FORTRAN.¹

ISO / IEC 646: 1991

ISO / IEC 646: 1991, Information Processing — ISO 7-Bit Coded Character Set for Information Interchange.²

ISO 4217: 2001

ISO 4217: 2001, Codes for the Representation of Currencies and Funds.

ISO 8601: 2000

ISO 8601: 2000, Data Elements and Interchange Formats — Information Interchange —

1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018, U.S.A.

2. ISO / IEC documents can be obtained from the ISO office: 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland / Suisse

161 Representation of Dates and Times.
 162 ISO C (1999)
 163 ISO/IEC 9899: 1999, Programming Languages — C, including Technical Corrigendum 1. |
 164 ISO/IEC 10646-1:2000
 165 ISO/IEC 10646-1:2000, Information Technology — Universal Multiple-Octet Coded
 166 Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

167 1.4 Terminology

168 For the purposes of IEEE Std 1003.1-2001, the following terminology definitions apply:

169 **can**

170 Describes a permissible optional feature or behavior available to the user or application. The
 171 feature or behavior is mandatory for an implementation that conforms to
 172 IEEE Std 1003.1-2001. An application can rely on the existence of the feature or behavior.

173 **implementation-defined**

174 Describes a value or behavior that is not defined by IEEE Std 1003.1-2001 but is selected by
 175 an implementor. The value or behavior may vary among implementations that conform to
 176 IEEE Std 1003.1-2001. An application should not rely on the existence of the value or
 177 behavior. An application that relies on such a value or behavior cannot be assured to be
 178 portable across conforming implementations.

179 The implementor shall document such a value or behavior so that it can be used correctly
 180 by an application.

181 **legacy**

182 Describes a feature or behavior that is being retained for compatibility with older
 183 applications, but which has limitations which make it inappropriate for developing portable
 184 applications. New applications should use alternative means of obtaining equivalent
 185 functionality.

186 **may**

187 Describes a feature or behavior that is optional for an implementation that conforms to
 188 IEEE Std 1003.1-2001. An application should not rely on the existence of the feature or
 189 behavior. An application that relies on such a feature or behavior cannot be assured to be
 190 portable across conforming implementations.

191 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

192 **shall**

193 For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or
 194 behavior that is mandatory. An application can rely on the existence of the feature or
 195 behavior.

196 For an application or user, describes a behavior that is mandatory.

197 **should**

198 For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or
 199 behavior that is recommended but not mandatory. An application should not rely on the
 200 existence of the feature or behavior. An application that relies on such a feature or behavior
 201 cannot be assured to be portable across conforming implementations.

202 For an application, describes a feature or behavior that is recommended programming
 203 practice for optimum portability.

undefined

Describes the nature of a value or behavior not defined by IEEE Std 1003.1-2001 which results from use of an invalid program construct or invalid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

unspecified

Describes the nature of a value or behavior not specified by IEEE Std 1003.1-2001 which results from use of a valid program construct or valid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

1.5 Portability

Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 and functions in the System Interfaces volume of IEEE Std 1003.1-2001 describe functionality that might not be fully portable to systems meeting the requirements for POSIX conformance (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance).

Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in the margin identifies the nature of the option, extension, or warning (see Section 1.5.1). For maximum portability, an application should avoid such functionality.

Unless the primary task of a utility is to produce textual material on its standard output, application developers should not rely on the format or content of any such material that may be produced. Where the primary task is to provide such material, but the output format is incompletely specified, the description is marked with the OF margin code and shading. Application developers are warned not to expect that the output of such an interface on one system is any guide to its behavior on another system.

1.5.1 Codes

The codes and their meanings are as follows. See also Section 1.5.2 (on page 14).

ADV

Advisory Information

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the ADV margin legend.

AIO

Asynchronous Input and Output

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the AIO margin legend.

246	BAR	Barriers
247		The functionality described is optional. The functionality described is also an extension to the
248		ISO C standard.
249		Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
250		Where additional semantics apply to a function, the material is identified by use of the BAR
251		margin legend.
252	BE	Batch Environment Services and Utilities
253		The functionality described is optional.
254		Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
255		Where additional semantics apply to a utility, the material is identified by use of the BE margin
256		legend.
257	CD	C-Language Development Utilities
258		The functionality described is optional.
259		Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
260		Where additional semantics apply to a utility, the material is identified by use of the CD margin
261		legend.
262	CPT	Process CPU-Time Clocks
263		The functionality described is optional. The functionality described is also an extension to the
264		ISO C standard.
265		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
266		Where additional semantics apply to a function, the material is identified by use of the CPT
267		margin legend.
268	CS	Clock Selection
269		The functionality described is optional. The functionality described is also an extension to the
270		ISO C standard.
271		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
272		Where additional semantics apply to a function, the material is identified by use of the CS
273		margin legend.
274	CX	Extension to the ISO C standard
275		The functionality described is an extension to the ISO C standard. Application writers may make
276		use of an extension as it is supported on all IEEE Std 1003.1-2001-conforming systems.
277		With each function or header from the ISO C standard, a statement to the effect that “any
278		conflict is unintentional” is included. That is intended to refer to a direct conflict.
279		IEEE Std 1003.1-2001 acts in part as a profile of the ISO C standard, and it may choose to further
280		constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered
281		conflicts.
282		Where additional semantics apply to a function or header, the material is identified by use of the
283		CX margin legend.
284	FD	FORTTRAN Development Utilities
285		The functionality described is optional.
286		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
287		Where additional semantics apply to a utility, the material is identified by use of the FD margin
288		legend.
289	FR	FORTTRAN Runtime Utilities
290		The functionality described is optional.

291		Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
292		Where additional semantics apply to a utility, the material is identified by use of the FR margin
293		legend.
294	FSC	File Synchronization
295		The functionality described is optional. The functionality described is also an extension to the
296		ISO C standard.
297		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
298		Where additional semantics apply to a function, the material is identified by use of the FSC
299		margin legend.
300	IP6	IPV6
301		The functionality described is optional. The functionality described is also an extension to the
302		ISO C standard.
303		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
304		Where additional semantics apply to a function, the material is identified by use of the IP6
305		margin legend.
306	MC1	Advisory Information and either Memory Mapped Files or Shared Memory Objects
307		The functionality described is optional. The functionality described is also an extension to the
308		ISO C standard.
309		This is a shorthand notation for combinations of multiple option codes.
310		Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
311		Where additional semantics apply to a function, the material is identified by use of the MC1
312		margin legend.
313		Refer to Section 1.5.2 (on page 14).
314	MC2	Memory Mapped Files, Shared Memory Objects, or Memory Protection
315		The functionality described is optional. The functionality described is also an extension to the
316		ISO C standard.
317		This is a shorthand notation for combinations of multiple option codes.
318		Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.
319		Where additional semantics apply to a function, the material is identified by use of the MC2
320		margin legend.
321		Refer to Section 1.5.2 (on page 14).
322	MC3	Memory Mapped Files, Shared Memory Objects, or Typed Memory Objects
323		The functionality described is optional. The functionality described is also an extension to the
324		ISO C standard.
325		This is a shorthand notation for combinations of multiple option codes.
326		Where applicable, functions are marked with the MC3 margin legend in the SYNOPSIS section.
327		Where additional semantics apply to a function, the material is identified by use of the MC3
328		margin legend.
329		Refer to Section 1.5.2 (on page 14).
330	MF	Memory Mapped Files
331		The functionality described is optional. The functionality described is also an extension to the
332		ISO C standard.

333 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.
 334 Where additional semantics apply to a function, the material is identified by use of the MF
 335 margin legend.

336 ML **Process Memory Locking**
 337 The functionality described is optional. The functionality described is also an extension to the
 338 ISO C standard.

339 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
 340 Where additional semantics apply to a function, the material is identified by use of the ML
 341 margin legend.

342 MLR **Range Memory Locking**
 343 The functionality described is optional. The functionality described is also an extension to the
 344 ISO C standard.

345 Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
 346 Where additional semantics apply to a function, the material is identified by use of the MLR
 347 margin legend.

348 MON **Monotonic Clock**
 349 The functionality described is optional. The functionality described is also an extension to the
 350 ISO C standard.

351 Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
 352 Where additional semantics apply to a function, the material is identified by use of the MON
 353 margin legend.

354 MPR **Memory Protection**
 355 The functionality described is optional. The functionality described is also an extension to the
 356 ISO C standard.

357 Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
 358 Where additional semantics apply to a function, the material is identified by use of the MPR
 359 margin legend.

360 MSG **Message Passing**
 361 The functionality described is optional. The functionality described is also an extension to the
 362 ISO C standard.

363 Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
 364 Where additional semantics apply to a function, the material is identified by use of the MSG
 365 margin legend.

366 MX **IEC 60559 Floating-Point Option**
 367 The functionality described is optional. The functionality described is also an extension to the
 368 ISO C standard.

369 Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.
 370 Where additional semantics apply to a function, the material is identified by use of the MX
 371 margin legend.

372 OB **Obsolescent**
 373 The functionality described may be withdrawn in a future version of this volume of
 374 IEEE Std 1003.1-2001. Strictly Conforming POSIX Applications and Strictly Conforming XSI
 375 Applications shall not use obsolescent features.

376 Where applicable, the material is identified by use of the OB margin legend.

377	OF	Output Format Incompletely Specified
378		The functionality described is an XSI extension. The format of the output produced by the utility
379		is not fully specified. It is therefore not possible to post-process this output in a consistent
380		fashion. Typical problems include unknown length of strings and unspecified field delimiters.
381		Where applicable, the material is identified by use of the OF margin legend.
382	OH	Optional Header
383		In the SYNOPSIS section of some interfaces in the System Interfaces volume of
384		IEEE Std 1003.1-2001 an included header is marked as in the following example:
385	OH	<code>#include <sys/types.h></code>
386		<code>#include <grp.h></code>
387		<code>struct group *getgrnam(const char *name);</code>
388		The OH margin legend indicates that the marked header is not required on XSI-conformant
389		systems.
390	PIO	Prioritized Input and Output
391		The functionality described is optional. The functionality described is also an extension to the
392		ISO C standard.
393		Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
394		Where additional semantics apply to a function, the material is identified by use of the PIO
395		margin legend.
396	PS	Process Scheduling
397		The functionality described is optional. The functionality described is also an extension to the
398		ISO C standard.
399		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
400		Where additional semantics apply to a function, the material is identified by use of the PS
401		margin legend.
402	RS	Raw Sockets
403		The functionality described is optional. The functionality described is also an extension to the
404		ISO C standard.
405		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
406		Where additional semantics apply to a function, the material is identified by use of the RS
407		margin legend.
408	RTS	Realtime Signals Extension
409		The functionality described is optional. The functionality described is also an extension to the
410		ISO C standard.
411		Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
412		Where additional semantics apply to a function, the material is identified by use of the RTS
413		margin legend.
414	SD	Software Development Utilities
415		The functionality described is optional.
416		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
417		Where additional semantics apply to a utility, the material is identified by use of the SD margin
418		legend.
419	SEM	Semaphores
420		The functionality described is optional. The functionality described is also an extension to the
421		ISO C standard.

422		Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
423		Where additional semantics apply to a function, the material is identified by use of the SEM
424		margin legend.
425	SHM	Shared Memory Objects
426		The functionality described is optional. The functionality described is also an extension to the
427		ISO C standard.
428		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
429		Where additional semantics apply to a function, the material is identified by use of the SHM
430		margin legend.
431	SIO	Synchronized Input and Output
432		The functionality described is optional. The functionality described is also an extension to the
433		ISO C standard.
434		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
435		Where additional semantics apply to a function, the material is identified by use of the SIO
436		margin legend.
437	SPI	Spin Locks
438		The functionality described is optional. The functionality described is also an extension to the
439		ISO C standard.
440		Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.
441		Where additional semantics apply to a function, the material is identified by use of the SPI
442		margin legend.
443	SPN	Spawn
444		The functionality described is optional. The functionality described is also an extension to the
445		ISO C standard.
446		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
447		Where additional semantics apply to a function, the material is identified by use of the SPN
448		margin legend.
449	SS	Process Sporadic Server
450		The functionality described is optional. The functionality described is also an extension to the
451		ISO C standard.
452		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
453		Where additional semantics apply to a function, the material is identified by use of the SS
454		margin legend.
455	TCT	Thread CPU-Time Clocks
456		The functionality described is optional. The functionality described is also an extension to the
457		ISO C standard.
458		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
459		Where additional semantics apply to a function, the material is identified by use of the TCT
460		margin legend.
461	TEF	Trace Event Filter
462		The functionality described is optional. The functionality described is also an extension to the
463		ISO C standard.
464		Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
465		Where additional semantics apply to a function, the material is identified by use of the TEF
466		margin legend.

467	THR	Threads
468		The functionality described is optional. The functionality described is also an extension to the
469		ISO C standard.
470		Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.
471		Where additional semantics apply to a function, the material is identified by use of the THR
472		margin legend.
473	TMO	Timeouts
474		The functionality described is optional. The functionality described is also an extension to the
475		ISO C standard.
476		Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
477		Where additional semantics apply to a function, the material is identified by use of the TMO
478		margin legend.
479	TMR	Timers
480		The functionality described is optional. The functionality described is also an extension to the
481		ISO C standard.
482		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
483		Where additional semantics apply to a function, the material is identified by use of the TMR
484		margin legend.
485	TPI	Thread Priority Inheritance
486		The functionality described is optional. The functionality described is also an extension to the
487		ISO C standard.
488		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
489		Where additional semantics apply to a function, the material is identified by use of the TPI
490		margin legend.
491	TPP	Thread Priority Protection
492		The functionality described is optional. The functionality described is also an extension to the
493		ISO C standard.
494		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
495		Where additional semantics apply to a function, the material is identified by use of the TPP
496		margin legend.
497	TPS	Thread Execution Scheduling
498		The functionality described is optional. The functionality described is also an extension to the
499		ISO C standard.
500		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
501		Where additional semantics apply to a function, the material is identified by use of the TPS
502		margin legend.
503	TRC	Trace
504		The functionality described is optional. The functionality described is also an extension to the
505		ISO C standard.
506		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
507		Where additional semantics apply to a function, the material is identified by use of the TRC
508		margin legend.
509	TRI	Trace Inherit
510		The functionality described is optional. The functionality described is also an extension to the
511		ISO C standard.

512		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
513		Where additional semantics apply to a function, the material is identified by use of the TRI
514		margin legend.
515	TRL	Trace Log
516		The functionality described is optional. The functionality described is also an extension to the
517		ISO C standard.
518		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
519		Where additional semantics apply to a function, the material is identified by use of the TRL
520		margin legend.
521	TSA	Thread Stack Address Attribute
522		The functionality described is optional. The functionality described is also an extension to the
523		ISO C standard.
524		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
525		Where additional semantics apply to a function, the material is identified by use of the TSA
526		margin legend.
527	TSF	Thread-Safe Functions
528		The functionality described is optional. The functionality described is also an extension to the
529		ISO C standard.
530		Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.
531		Where additional semantics apply to a function, the material is identified by use of the TSF
532		margin legend.
533	TSH	Thread Process-Shared Synchronization
534		The functionality described is optional. The functionality described is also an extension to the
535		ISO C standard.
536		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
537		Where additional semantics apply to a function, the material is identified by use of the TSH
538		margin legend.
539	TSP	Thread Sporadic Server
540		The functionality described is optional. The functionality described is also an extension to the
541		ISO C standard.
542		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
543		Where additional semantics apply to a function, the material is identified by use of the TSP
544		margin legend.
545	TSS	Thread Stack Size Attribute
546		The functionality described is optional. The functionality described is also an extension to the
547		ISO C standard.
548		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
549		Where additional semantics apply to a function, the material is identified by use of the TSS
550		margin legend.
551	TYM	Typed Memory Objects
552		The functionality described is optional. The functionality described is also an extension to the
553		ISO C standard.
554		Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
555		Where additional semantics apply to a function, the material is identified by use of the TYM
556		margin legend.

557 UP User Portability Utilities

558 The functionality described is optional.

559 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
560 Where additional semantics apply to a utility, the material is identified by use of the UP margin
561 legend.

562 XSI Extension

563 The functionality described is an XSI extension. Functionality marked XSI is also an extension to
564 the ISO C standard. Application writers may confidently make use of an extension on all
565 systems supporting the X/Open System Interfaces Extension.

566 If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that
567 reference page is an extension. See Section 2.1.4 (on page 21).

568 XSR XSI STREAMS

569 The functionality described is optional. The functionality described is also an extension to the
570 ISO C standard.

571 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.
572 Where additional semantics apply to a function, the material is identified by use of the XSR
573 margin legend.

574 1.5.2 Margin Code Notation

575 Some of the functionality described in IEEE Std 1003.1-2001 depends on support of more than
576 one option, or independently may depend on several options. The following notation for margin
577 codes is used to denote the following cases.

578 A Feature Dependent on One or Two Options

579 In this case, margin codes have a <space> separator; for example:

580 MF This feature requires support for only the Memory Mapped Files option.

581 MF SHM This feature requires support for both the Memory Mapped Files and the Shared Memory
582 Objects options; that is, an application which uses this feature is portable only between
583 implementations that provide both options.

584 A Feature Dependent on Either of the Options Denoted

585 In this case, margin codes have a ' | ' separator to denote the logical OR; for example:

586 MF|SHM This feature is dependent on support for either the Memory Mapped Files option or the Shared
587 Memory Objects option; that is, an application which uses this feature is portable between
588 implementations that provide any (or all) of the options.

589 A Feature Dependent on More than Two Options

590 The following shorthand notations are used:

591 MC1 The MC1 margin code is shorthand for ADV (MF|SHM). Features which are shaded with this
592 margin code require support of the Advisory Information option and either the Memory
593 Mapped Files or Shared Memory Objects option.

594 MC2 The MC2 margin code is shorthand for MF|SHM|MPR. Features which are shaded with this
595 margin code require support of either the Memory Mapped Files, Shared Memory Objects, or
596 Memory Protection options.

597 MC3 The MC3 margin code is shorthand for MF|SHM|TYM. Features which are shaded with this |
598 margin code require support of either the Memory Mapped Files, Shared Memory Objects, or |
599 Typed Memory Objects options. |

600 **Large Sections Dependent on an Option**

601 Where large sections of text are dependent on support for an option, a lead-in text block is
602 provided and shaded accordingly; for example:

603 TRC This section describes extensions to support tracing of user applications. This functionality is
604 dependent on support of the Trace option (and the rest of this section is not further shaded for
605 this option).

2.1 Implementation Conformance

2.1.1 Requirements

A *conforming implementation* shall meet all of the following criteria:

1. The system shall support all utilities, functions, and facilities defined within IEEE Std 1003.1-2001 that are required for POSIX conformance (see Section 2.1.3 (on page 18)). These interfaces shall support the functional behavior described herein.
2. The system may support one or more options as described under Section 2.1.5 (on page 22). When an implementation claims that an option is supported, all of its constituent parts shall be provided.
3. The system may support the X/Open System Interface Extension (XSI) as described under Section 2.1.4 (on page 21).
4. The system may provide additional utilities, functions, or facilities not required by IEEE Std 1003.1-2001. Non-standard extensions of the utilities, functions, or facilities specified in IEEE Std 1003.1-2001 should be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of utilities, functions, or facilities defined by IEEE Std 1003.1-2001. The conformance document shall define an environment in which an application can be run with the behavior specified by IEEE Std 1003.1-2001. In no case shall such an environment require modification of a Strictly Conforming POSIX Application (see Section 2.2.1 (on page 31)).

2.1.2 Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to IEEE Std 1003.1-2001. The conformance document shall have the same structure as IEEE Std 1003.1-2001, with the information presented in the appropriate sections and subsections. Sections and subsections that consist solely of subordinate section titles, with no other information, are not required. The conformance document shall not contain information about extended facilities or capabilities outside the scope of IEEE Std 1003.1-2001.

The conformance document shall contain a statement that indicates the full name, number, and date of the standard that applies. The conformance document may also list international software standards that are available for use by a Conforming POSIX Application. Applicable characteristics where documentation is required by one of these standards, or by standards of government bodies, may also be included.

The conformance document shall describe the limit values found in the headers **<limits.h>** (on page 249) and **<unistd.h>** (on page 400), stating values, the conditions under which those values may change, and the limits of such variations, if any.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in IEEE Std 1003.1-2001. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. When the value or behavior in the

implementation is designed to be variable or customized on each instantiation of the system, the implementation provider shall document the nature and permissible ranges of this variation.

The conformance document may specify the behavior of the implementation for those features where IEEE Std 1003.1-2001 states that implementations may vary or where features are identified as undefined or unspecified.

The conformance document shall not contain documentation other than that specified in the preceding paragraphs except where such documentation is specifically allowed or required by other provisions of IEEE Std 1003.1-2001.

The phrases “shall document” or “shall be documented” in IEEE Std 1003.1-2001 mean that documentation of the feature shall appear in the conformance document, as described previously, unless there is an explicit reference in the conformance document to show where the information can be found in the system documentation.

The system documentation should also contain the information found in the conformance document.

2.1.3 POSIX Conformance

A conforming implementation shall meet the following criteria for POSIX conformance.

2.1.3.1 POSIX System Interfaces

- The system shall support all the mandatory functions and headers defined in IEEE Std 1003.1-2001, and shall set the symbolic constant `_POSIX_VERSION` to the value 200112L.
- Although all implementations conforming to IEEE Std 1003.1-2001 support all the features described below, there may be system-dependent or file system-dependent configuration procedures that can remove or modify any or all of these features. Such configurations should not be made if strict compliance is required.

The following symbolic constants shall either be undefined or defined with a value other than `-1`. If a constant is undefined, an application should use the `sysconf()`, `pathconf()`, or `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the system at that time or for the particular pathname in question.

— `_POSIX_CHOWN_RESTRICTED`

The use of `chown()` is restricted to a process with appropriate privileges, and to changing the group ID of a file only to the effective group ID of the process or to one of its supplementary group IDs.

— `_POSIX_NO_TRUNC`

Pathname components longer than `{NAME_MAX}` generate an error.

- The following symbolic constants shall be defined as follows:
 - `_POSIX_JOB_CONTROL` shall have a value greater than zero.
 - `_POSIX_SAVED_IDS` shall have a value greater than zero.
 - `_POSIX_VDISABLE` shall have a value other than `-1`.

Note: The symbols above represent historical options that are no longer allowed as options, but are retained here for backwards-compatibility of applications.

- 686 • The system may support one or more options (see Section 2.1.6 (on page 28)) denoted by the
687 following symbolic constants:
- 688 — _POSIX_ADVISORY_INFO
689 — _POSIX_ASYNCHRONOUS_IO
690 — _POSIX_BARRIERS
691 — _POSIX_CLOCK_SELECTION
692 — _POSIX_CPUTIME
693 — _POSIX_FSYNC
694 — _POSIX_IPV6
695 — _POSIX_MAPPED_FILES
696 — _POSIX_MEMLOCK
697 — _POSIX_MEMLOCK_RANGE
698 — _POSIX_MEMORY_PROTECTION
699 — _POSIX_MESSAGE_PASSING
700 — _POSIX_MONOTONIC_CLOCK
701 — _POSIX_PRIORITIZED_IO
702 — _POSIX_PRIORITY_SCHEDULING
703 — _POSIX_RAW_SOCKETS
704 — _POSIX_REALTIME_SIGNALS
705 — _POSIX_SEMAPHORES
706 — _POSIX_SHARED_MEMORY_OBJECTS
707 — _POSIX_SPAWN
708 — _POSIX_SPIN_LOCKS
709 — _POSIX_SPORADIC_SERVER
710 — _POSIX_SYNCHRONIZED_IO
711 — _POSIX_THREAD_ATTR_STACKADDR
712 — _POSIX_THREAD_CPUTIME
713 — _POSIX_THREAD_ATTR_STACKSIZE
714 — _POSIX_THREAD_PRIO_INHERIT
715 — _POSIX_THREAD_PRIO_PROTECT
716 — _POSIX_THREAD_PRIORITY_SCHEDULING
717 — _POSIX_THREAD_PROCESS_SHARED
718 — _POSIX_THREAD_SAFE_FUNCTIONS
719 — _POSIX_THREAD_SPORADIC_SERVER
720 — _POSIX_THREADS

721	—	_POSIX_TIMEOUTS
722	—	_POSIX_TIMERS
723	—	_POSIX_TRACE
724	—	_POSIX_TRACE_EVENT_FILTER
725	—	_POSIX_TRACE_INHERIT
726	—	_POSIX_TRACE_LOG
727	—	_POSIX_TYPED_MEMORY_OBJECTS
728	If any of the symbolic constants _POSIX_TRACE_EVENT_FILTER, _POSIX_TRACE_LOG, or	
729	_POSIX_TRACE_INHERIT is defined to have a value other than -1, then the symbolic	
730	constant _POSIX_TRACE shall also be defined to have a value other than -1.	
731	XSI	• The system may support the XSI extensions (see Section 2.1.5.2 (on page 24)) denoted by the
732		following symbolic constants:
733	—	_XOPEN_CRYPT
734	—	_XOPEN_LEGACY
735	—	_XOPEN_REALTIME
736	—	_XOPEN_REALTIME_THREADS
737	—	_XOPEN_UNIX
738	2.1.3.2	<i>POSIX Shell and Utilities</i>
739	•	The system shall provide all the mandatory utilities in the Shell and Utilities volume of
740		IEEE Std 1003.1-2001 with all the functional behavior described therein.
741	•	The system shall support the Large File capabilities described in the Shell and Utilities
742		volume of IEEE Std 1003.1-2001.
743	•	The system may support one or more options (see Section 2.1.6 (on page 28)) denoted by the
744		following symbolic constants. (The literal names below apply to the <i>getconf</i> utility.)
745	—	POSIX2_C_DEV
746	—	POSIX2_CHAR_TERM
747	—	POSIX2_FORT_DEV
748	—	POSIX2_FORT_RUN
749	—	POSIX2_LOCALEDEF
750	—	POSIX2_PBS
751	—	POSIX2_PBS_ACCOUNTING
752	—	POSIX2_PBS_LOCATE
753	—	POSIX2_PBS_MESSAGE
754	—	POSIX2_PBS_TRACK
755	—	POSIX2_SW_DEV
756	—	POSIX2_UPE

- The system may support the XSI extensions (see Section 2.1.4).

Additional language bindings and development utility options may be provided in other related standards or in a future version of IEEE Std 1003.1-2001. In the former case, additional symbolic constants of the same general form as shown in this subsection should be defined by the related standard document and made available to the application without requiring IEEE Std 1003.1-2001 to be updated.

2.1.4 XSI Conformance

XSI This section describes the criteria for implementations conforming to the XSI extension (see Section 3.439 (on page 96)). This functionality is dependent on the support of the XSI extension (and the rest of this section is not further shaded).

IEEE Std 1003.1-2001 describes utilities, functions, and facilities offered to application programs by the X/Open System Interface (XSI). An XSI-conforming implementation shall meet the criteria for POSIX conformance and the following requirements.

2.1.4.1 XSI System Interfaces

- The system shall support all the functions and headers defined in IEEE Std 1003.1-2001 as part of the XSI extension denoted by the symbolic constant `_XOPEN_UNIX` and any extensions marked with the XSI extension marking (see Section 1.5.1 (on page 6)).
- The system shall support the `mmap()`, `munmap()`, and `msync()` functions.
- The system shall support the following options defined within IEEE Std 1003.1-2001 (see Section 2.1.6 (on page 28)):
 - `_POSIX_FSYNC`
 - `_POSIX_MAPPED_FILES`
 - `_POSIX_MEMORY_PROTECTION`
 - `_POSIX_THREAD_ATTR_STACKADDR`
 - `_POSIX_THREAD_ATTR_STACKSIZE`
 - `_POSIX_THREAD_PROCESS_SHARED`
 - `_POSIX_THREAD_SAFE_FUNCTIONS`
 - `_POSIX_THREADS`
- The system may support the following XSI Option Groups (see Section 2.1.5.2 (on page 24)) defined within IEEE Std 1003.1-2001:
 - Encryption
 - Realtime
 - Advanced Realtime
 - Realtime Threads
 - Advanced Realtime Threads
 - Tracing
 - XSI STREAMS
 - Legacy

2.1.4.2 XSI Shell and Utilities Conformance

- The system shall support all the utilities defined in the Shell and Utilities volume of IEEE Std 1003.1-2001 as part of the XSI extension denoted by the XSI marking in the SYNOPSIS section, and any extensions marked with the XSI extension marking (see Section 1.5.1 (on page 6)) within the text.
 - The system shall support the User Portability Utilities option.
 - The system shall support creation of locales (see Chapter 7 (on page 123)).
 - The C-language Development utility `c99` shall be supported.
 - The XSI Development Utilities option may be supported. It consists of the following software development utilities:
- | | | | | |
|--------------------|--------------------|---------------------|--------------------|--|
| <code>admin</code> | <code>delta</code> | <code>prs</code> | <code>unget</code> | |
| <code>cflow</code> | <code>get</code> | <code>rm del</code> | <code>val</code> | |
| <code>ctags</code> | <code>m4</code> | <code>sact</code> | <code>what</code> | |
| <code>cxref</code> | <code>nm</code> | <code>sccs</code> | | |
- Within the utilities that are provided, functionality marked by the code OF (see Section 1.5.1 (on page 6)) need not be provided.

2.1.5 Option Groups

An Option Group is a group of related functions or options defined within the System Interfaces volume of IEEE Std 1003.1-2001.

If an implementation supports an Option Group, then the system shall support the functional behavior described herein.

If an implementation does not support an Option Group, then the system need not support the functional behavior described herein.

2.1.5.1 Subprofiling Considerations

Profiling standards supporting functional requirements less than that required in IEEE Std 1003.1-2001 may subset both mandatory and optional functionality required for POSIX Conformance (see Section 2.1.3 (on page 18)) or XSI Conformance (see Section 2.1.4 (on page 21)). Such profiles shall organize the subsets into Subprofiling Option Groups.

The Rationale (Informative) volume of IEEE Std 1003.1-2001, Appendix E, Subprofiling Considerations (Informative) describes a representative set of such Subprofiling Option Groups for use by profiles applicable to specialized realtime systems. IEEE Std 1003.1-2001 does not require that the presence of Subprofiling Option Groups be testable at compile-time (as symbols defined in any header) or at runtime (via `sysconf()` or `getconf()`).

A Subprofiling Option Group may provide basic system functionality that other Subprofiling Option Groups and other options depend upon.³ If a profile of IEEE Std 1003.1-2001 does not

3. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation which are needed by any interface in IEEE Std 1003.1-2001 that parses a *path* argument. If a profile requires support for the Device Input and Output profiling option group but does not require support for the File System profiling option group, the profile must specify how pathname resolution is to behave in that profile, how the `O_CREAT` flag to `open()` is to be handled (and the use of the character 'a' in the *mode* argument of `open()` when a filename argument names a file that does not exist), and specify lots of other details.

require an implementation to provide a Subprofiling Option Group that provides features utilized by a required Subprofiling Option Group (or option),⁴ the profile shall specify⁵ all of the following:

- Restricted or altered behavior of interfaces defined in IEEE Std 1003.1-2001 that may differ on an implementation of the profile
- Additional behaviors that may produce undefined or unspecified results
- Additional implementation-defined behavior that implementations shall be required to document in the profile's conformance document

if any of the above is a result of the profile not requiring an interface required by IEEE Std 1003.1-2001.

The following additional rules shall apply to all profiles of IEEE Std 1003.1-2001:

- Any application that conforms to that profile shall also conform to IEEE Std 1003.1-2001 (that is, a profile shall not require restricted, altered, or extended behaviors of an implementation of IEEE Std 1003.1-2001).
- Profiles are permitted to add additional requirements to the limits defined in **<limits.h>** and **<stdint.h>**, subject to the following:
 - For the limits in **<limits.h>** and **<stdint.h>**:
 - If the limit is specified as having a fixed value, it shall not be changed by a profile.
 - If a limit is specified as having a minimum or maximum acceptable value, it may be changed by a profile as follows:
 - A profile may increase a minimum acceptable value, but shall not make a minimum acceptable value smaller.
 - A profile may reduce a maximum acceptable value, but shall not make a maximum acceptable value larger.
 - A profile shall not change a limit specified as having a minimum or maximum value into a limit specified as having a fixed value.
 - A profile shall not create new limits.
 - Any implementation that conforms to IEEE Std 1003.1-2001 (including all options and extended limits required by the profile) shall also conform to that profile.

4. As an example, IEEE Std 1003.1-2001 requires that implementations claiming to support the Range Memory Locking option also support the Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied without requiring that the Process Memory Locking option be supplied as long as the profile specifies everything an application writer or system implementor would have to know to build an application or implementation conforming to the profile.

5. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or unspecified results.

873 2.1.5.2 XSI Option Groups

874 xsi This section describes Option Groups to support the definition of XSI conformance within the
 875 System Interfaces volume of IEEE Std 1003.1-2001. This functionality is dependent on the
 876 support of the XSI extension (and the rest of this section is not further shaded).

877 The following Option Groups are defined.

878 **Encryption**

879 The Encryption Option Group is denoted by the symbolic constant `_XOPEN_CRYPT`. It includes
 880 the following functions:

881 `crypt()`, `encrypt()`, `setkey()`

882 These functions are marked CRYPT.

883 Due to export restrictions on the decoding algorithm in some countries, implementations may be
 884 restricted in making these functions available. All the functions in the Encryption Option Group
 885 may therefore return `[ENOSYS]` or, alternatively, `encrypt()` shall return `[ENOSYS]` for the
 886 decryption operation.

887 An implementation that claims conformance to this Option Group shall set `_XOPEN_CRYPT` to
 888 a value other than `-1`.

889 **Realtime**

890 The Realtime Option Group is denoted by the symbolic constant `_XOPEN_REALTIME`.

891 This Option Group includes a set of realtime functions drawn from options within
 892 IEEE Std 1003.1-2001 (see Section 2.1.6 (on page 28)).

893 Where entire functions are included in the Option Group, the NAME section is marked with
 894 REALTIME. Where additional semantics have been added to existing pages, the new material is
 895 identified by use of the appropriate margin legend for the underlying option defined within
 896 IEEE Std 1003.1-2001.

897 An implementation that claims conformance to this Option Group shall set
 898 `_XOPEN_REALTIME` to a value other than `-1`.

899 This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001
 900 (see Section 2.1.6 (on page 28)):

901 `_POSIX_ASYNCHRONOUS_IO`
 902 `_POSIX_FSYNC`
 903 `_POSIX_MAPPED_FILES`
 904 `_POSIX_MEMLOCK`
 905 `_POSIX_MEMLOCK_RANGE`
 906 `_POSIX_MEMORY_PROTECTION`
 907 `_POSIX_MESSAGE_PASSING`
 908 `_POSIX_PRIORITIZED_IO`
 909 `_POSIX_PRIORITY_SCHEDULING`
 910 `_POSIX_REALTIME_SIGNALS`
 911 `_POSIX_SEMAPHORES`
 912 `_POSIX_SHARED_MEMORY_OBJECTS`
 913 `_POSIX_SYNCHRONIZED_IO`
 914 `_POSIX_TIMERS`

If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the following symbolic constants shall be defined by the implementation to have the value 200112L:

```

_POSIX_ASYNCHRONOUS_IO
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MESSAGE_PASSING
_POSIX_PRIORITY_SCHEDULING
_POSIX_REALTIME_SIGNALS
_POSIX_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO
_POSIX_TIMERS

```

The functionality associated with `_POSIX_MAPPED_FILES`, `_POSIX_MEMORY_PROTECTION`, and `_POSIX_FSYNC` is always supported on XSI-conformant systems.

Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If this functionality is supported, then `_POSIX_PRIORITIZED_IO` shall be set to a value other than `-1`. Otherwise, it shall be undefined.

If `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling priority of the process minus `aiocbp->aio_reqprio`. The implementation shall also document for which files I/O prioritization is supported.

Advanced Realtime

An implementation that claims conformance to this Option Group shall also support the Realtime Option Group.

Where entire functions are included in the Option Group, the NAME section is marked with **ADVANCED REALTIME**. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within IEEE Std 1003.1-2001.

This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001 (see Section 2.1.6 (on page 28)):

```

_POSIX_ADVISORY_INFO
_POSIX_CLOCK_SELECTION
_POSIX_CPUTIME
_POSIX_MONOTONIC_CLOCK
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_TIMEOUTS
_POSIX_TYPED_MEMORY_OBJECTS

```

If the implementation supports the Advanced Realtime Option Group, then the following symbolic constants shall be defined by the implementation to have the value 200112L:

955 _POSIX_ADVISORY_INFO
 956 _POSIX_CLOCK_SELECTION
 957 _POSIX_CPUTIME
 958 _POSIX_MONOTONIC_CLOCK
 959 _POSIX_SPAWN
 960 _POSIX_SPORADIC_SERVER
 961 _POSIX_TIMEOUTS
 962 _POSIX_TYPED_MEMORY_OBJECTS

963 If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant
 964 `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the
 965 value 200112L.

966 If the symbolic constant `_POSIX_CPUTIME` is defined, then the symbolic constant
 967 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200112L.

968 If the symbolic constant `_POSIX_MONOTONIC_CLOCK` is defined, then the symbolic constant
 969 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200112L.

970 If the symbolic constant `_POSIX_CLOCK_SELECTION` is defined, then the symbolic constant
 971 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200112L.

972 **Realtime Threads**

973 The Realtime Threads Option Group is denoted by the symbolic constant
 974 `_XOPEN_REALTIME_THREADS`.

975 This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001
 976 (see Section 2.1.6 (on page 28)):

977 _POSIX_THREAD_PRIO_INHERIT
 978 _POSIX_THREAD_PRIO_PROTECT
 979 _POSIX_THREAD_PRIORITY_SCHEDULING

980 Where applicable, whole pages are marked `REALTIME THREADS`, together with the
 981 appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 6)).

982 An implementation that claims conformance to this Option Group shall set
 983 `_XOPEN_REALTIME_THREADS` to a value other than `-1`.

984 If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the
 985 following options shall also be defined by the implementation to have the value 200112L:

986 _POSIX_THREAD_PRIO_INHERIT
 987 _POSIX_THREAD_PRIO_PROTECT
 988 _POSIX_THREAD_PRIORITY_SCHEDULING

989 **Advanced Realtime Threads**

990 An implementation that claims conformance to this Option Group shall also support the
 991 Realtime Threads Option Group.

992 Where entire functions are included in the Option Group, the NAME section is marked with
 993 `ADVANCED REALTIME THREADS`. Where additional semantics have been added to existing
 994 pages, the new material is identified by use of the appropriate margin legend for the underlying
 995 option defined within IEEE Std 1003.1-2001.

996 This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001
 997 (see Section 2.1.6 (on page 28)):

998 _POSIX_BARRIERS
 999 _POSIX_SPIN_LOCKS
 1000 _POSIX_THREAD_CPUTIME
 1001 _POSIX_THREAD_SPORADIC_SERVER

1002 If the symbolic constant _POSIX_THREAD_SPORADIC_SERVER is defined to have the value
 1003 200112L, then the symbolic constant _POSIX_THREAD_PRIORITY_SCHEDULING shall also be
 1004 defined by the implementation to have the value 200112L.

1005 If the symbolic constant _POSIX_THREAD_CPUTIME is defined to have the value 200112L,
 1006 then the symbolic constant _POSIX_TIMERS shall also be defined by the implementation to have
 1007 the value 200112L.

1008 If the symbolic constant _POSIX_BARRIERS is defined to have the value 200112L, then the
 1009 symbolic constants _POSIX_THREADS and _POSIX_THREAD_SAFE_FUNCTIONS shall also
 1010 be defined by the implementation to have the value 200112L.

1011 If the symbolic constant _POSIX_SPIN_LOCKS is defined to have the value 200112L, then the
 1012 symbolic constants _POSIX_THREADS and _POSIX_THREAD_SAFE_FUNCTIONS shall also
 1013 be defined by the implementation to have the value 200112L.

1014 If the implementation supports the Advanced Realtime Threads Option Group, then the
 1015 following symbolic constants shall be defined by the implementation to have the value 200112L:

1016 _POSIX_BARRIERS
 1017 _POSIX_SPIN_LOCKS
 1018 _POSIX_THREAD_CPUTIME
 1019 _POSIX_THREAD_SPORADIC_SERVER

1020 **Tracing**

1021 This Option Group includes a set of tracing functions drawn from options within
 1022 IEEE Std 1003.1-2001 (see Section 2.1.6 (on page 28)).

1023 Where entire functions are included in the Option Group, the NAME section is marked with
 1024 TRACING. Where additional semantics have been added to existing pages, the new material is
 1025 identified by use of the appropriate margin legend for the underlying option defined within
 1026 IEEE Std 1003.1-2001.

1027 This Option Group consists of the set of the following options from within IEEE Std 1003.1-2001
 1028 (see Section 2.1.6 (on page 28)):

1029 _POSIX_TRACE
 1030 _POSIX_TRACE_EVENT_FILTER
 1031 _POSIX_TRACE_LOG
 1032 _POSIX_TRACE_INHERIT

1033 If the implementation supports the Tracing Option Group, then the following symbolic
 1034 constants shall be defined by the implementation to have the value 200112L:

1035 _POSIX_TRACE
 1036 _POSIX_TRACE_EVENT_FILTER
 1037 _POSIX_TRACE_LOG
 1038 _POSIX_TRACE_INHERIT

XSI STREAMS

The XSI STREAMS Option Group is denoted by the symbolic constant `_XOPEN_STREAMS`.

This Option Group includes functionality related to STREAMS, a uniform mechanism for implementing networking services and other character-based I/O as described in the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.6, STREAMS.

It includes the following functions:

fattach(), *fdetach()*, *getmsg()*, *getpmsg()*, *ioctl()*, *isastream()*, *putmsg()*, *putpmsg()*

and the `<stropts.h>` header.

Where applicable, whole pages are marked STREAMS, together with the appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 6)). Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within IEEE Std 1003.1-2001.

An implementation that claims conformance to this Option Group shall set `_XOPEN_STREAMS` to a value other than `-1`.

Legacy

The Legacy Option Group is denoted by the symbolic constant `_XOPEN_LEGACY`.

The Legacy Option Group includes the functions and headers which were mandatory in previous versions of IEEE Std 1003.1-2001 but are optional in this version.

These functions and headers are retained in IEEE Std 1003.1-2001 because of their widespread use. Application writers should not rely on the existence of these functions or headers in new applications, but should follow the migration path detailed in the APPLICATION USAGE sections of the relevant pages.

Various factors may have contributed to the decision to mark a function or header LEGACY. In all cases, the specific reasons for the withdrawal of a function or header are documented on the relevant pages.

Once a function or header is marked LEGACY, no modifications are made to the specifications of such functions or headers other than to the APPLICATION USAGE sections of the relevant pages.

The functions and headers which form this Option Group are as follows:

bcmp(), *bcopy()*, *bzero()*, *ecvt()*, *fcvt()*, *ftime()*, *gcvt()*, *getwd()*, *index()*, *mktemp()*, *rindex()*, *utimes()*, *wcs wcs()*

An implementation that claims conformance to this Option Group shall set `_XOPEN_LEGACY` to a value other than `-1`.

2.1.6 Options

The symbolic constants defined in `<unistd.h>`, **Constants for Options and Option Groups** (on page 400) reflect implementation options for IEEE Std 1003.1-2001. These symbols can be used by the application to determine which optional facilities are present on the implementation. The *sysconf()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 or the *getconf* utility defined in the Shell and Utilities volume of IEEE Std 1003.1-2001 can be used to retrieve the value of each symbol on each specific implementation to determine whether the option is supported.

1080		Where an option is not supported, the associated utilities, functions, or facilities need not be
1081		present.
1082		Margin codes are defined for each option (see Section 1.5.1 (on page 6)).
1083	2.1.6.1	<i>System Interfaces</i>
1084		Refer to <unistd.h>, Constants for Options and Option Groups (on page 400) for the list of
1085		options.
1086	2.1.6.2	<i>Shell and Utilities</i>
1087		Each of these symbols shall be considered valid names by the implementation. Refer to
1088		<unistd.h>, Constants for Options and Option Groups (on page 400).
1089		The literal names shown below apply only to the <i>getconf</i> utility.
1090	CD	POSIX2_C_DEV
1091		The system supports the C-Language Development Utilities option.
1092		The utilities in the C-Language Development Utilities option are used for the development
1093		of C-language applications, including compilation or translation of C source code and
1094		complex program generators for simple lexical tasks and processing of context-free
1095		grammars.
1096		The utilities listed below may be provided by a conforming system; however, any system
1097		claiming conformance to the C-Language Development Utilities option shall provide all of
1098		the utilities listed.
1099		<i>c99</i>
1100		<i>lex</i>
1101		<i>yacc</i>
1102		POSIX2_CHAR_TERM
1103		The system supports the Terminal Characteristics option. This value need not be present on
1104		a system not supporting the User Portability Utilities option.
1105		Where applicable, the dependency is noted within the description of the utility.
1106		This option applies only to systems supporting the User Portability Utilities option. If
1107		supported, then the system supports at least one terminal type capable of all operations
1108		described in IEEE Std 1003.1-2001; see Section 10.2 (on page 185).
1109	FD	POSIX2_FORT_DEV
1110		The system supports the FORTRAN Development Utilities option.
1111		The <i>fort77</i> FORTRAN compiler is the only utility in the FORTRAN Development Utilities
1112		option. This is used for the development of FORTRAN language applications, including
1113		compilation or translation of FORTRAN source code.
1114		The <i>fort77</i> utility may be provided by a conforming system; however, any system claiming
1115		conformance to the FORTRAN Development Utilities option shall provide the <i>fort77</i> utility.
1116	FR	POSIX2_FORT_RUN
1117		The system supports the FORTRAN Runtime Utilities option.
1118		The <i>asa</i> utility is the only utility in the FORTRAN Runtime Utilities option.
1119		The <i>asa</i> utility may be provided by a conforming system; however, any system claiming
1120		conformance to the FORTRAN Runtime Utilities option shall provide the <i>asa</i> utility.

1121		POSIX2_LOCALEDEF
1122		The system supports the Locale Creation Utilities option.
1123		If supported, the system supports the creation of locales as described in the <i>localedef</i> utility.
1124		The <i>localedef</i> utility may be provided by a conforming system; however, any system
1125		claiming conformance to the Locale Creation Utilities option shall provide the <i>localedef</i>
1126		utility.
1127	BE	POSIX2_PBS
1128		The system supports the Batch Environment Services and Utilities option (see the Shell and
1129		Utilities volume of IEEE Std 1003.1-2001, Chapter 3, Batch Environment Services).
1130		Note: The Batch Environment Services and Utilities option is a combination of mandatory and
1131		optional batch services and utilities. The POSIX_PBS symbolic constant implies the
1132		system supports all the mandatory batch services and utilities.
1133		POSIX2_PBS_ACCOUNTING
1134		The system supports the Batch Accounting option.
1135		POSIX2_PBS_CHECKPOINT
1136		The system supports the Batch Checkpoint/Restart option.
1137		POSIX2_PBS_LOCATE
1138		The system supports the Locate Batch Job Request option.
1139		POSIX2_PBS_MESSAGE
1140		The system supports the Batch Job Message Request option.
1141		POSIX2_PBS_TRACK
1142		The system supports the Track Batch Job Request option.
1143	SD	POSIX2_SW_DEV
1144		The system supports the Software Development Utilities option.
1145		The utilities in the Software Development Utilities option are used for the development of
1146		applications, including compilation or translation of source code, the creation and
1147		maintenance of library archives, and the maintenance of groups of inter-dependent
1148		programs.
1149		The utilities listed below may be provided by the conforming system; however, any system
1150		claiming conformance to the Software Development Utilities option shall provide all of the
1151		utilities listed here.
1152		<i>ar</i>
1153		<i>make</i>
1154		<i>nm</i>
1155		<i>strip</i>
1156	UP	POSIX2_UPE
1157		The system supports the User Portability Utilities option.
1158		The utilities in the User Portability Utilities option shall be implemented on all systems that
1159		claim conformance to this option. Certain utilities are noted as having features that cannot
1160		be implemented on all terminal types; if the POSIX2_CHAR_TERM option is supported, the
1161		system shall support all such features on at least one terminal type; see Section 10.2 (on
1162		page 185).
1163		Some of the utilities are required only on systems that also support the Software
1164		Development Utilities option, or the character-at-a-time terminal option (see Section 10.2
1165		(on page 185)); such utilities have this noted in their DESCRIPTION sections. All of the

1166 other utilities listed are required only on systems that claim conformance to the User
1167 Portability Utilities option.

1168	<i>alias</i>	<i>expand</i>	<i>nm</i>	<i>unalias</i>
1169	<i>at</i>	<i>fc</i>	<i>patch</i>	<i>unexpand</i>
1170	<i>batch</i>	<i>fg</i>	<i>ps</i>	<i>uudecode</i>
1171	<i>bg</i>	<i>file</i>	<i>renice</i>	<i>uuencode</i>
1172	<i>crontab</i>	<i>jobs</i>	<i>split</i>	<i>vi</i>
1173	<i>split</i>	<i>man</i>	<i>strings</i>	<i>who</i>
1174	<i>ctags</i>	<i>mesg</i>	<i>tabs</i>	<i>write</i>
1175	<i>df</i>	<i>more</i>	<i>talk</i>	
1176	<i>du</i>	<i>newgrp</i>	<i>time</i>	
1177	<i>ex</i>	<i>nice</i>	<i>tput</i>	

1178 2.2 Application Conformance

1179 All applications claiming conformance to IEEE Std 1003.1-2001 shall use only language-
1180 dependent services for the C programming language described in Section 2.3 (on page 33), shall
1181 use only the utilities and facilities defined in the Shell and Utilities volume of
1182 IEEE Std 1003.1-2001, and shall fall within one of the following categories.

1183 2.2.1 Strictly Conforming POSIX Application

1184 A Strictly Conforming POSIX Application is an application that requires only the facilities
1185 described in IEEE Std 1003.1-2001. Such an application:

- 1186 1. Shall accept any implementation behavior that results from actions it takes in areas
1187 described in IEEE Std 1003.1-2001 as *implementation-defined* or *unspecified*, or where
1188 IEEE Std 1003.1-2001 indicates that implementations may vary
- 1189 2. Shall not perform any actions that are described as producing *undefined* results
- 1190 3. For symbolic constants, shall accept any value in the range permitted by
1191 IEEE Std 1003.1-2001, but shall not rely on any value in the range being greater than the
1192 minimums listed or being less than the maximums listed in IEEE Std 1003.1-2001
- 1193 4. Shall not use facilities designated as *obsolescent*
- 1194 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
1195 facilities whose availability is indicated by Section 2.1.3 (on page 18)
- 1196 6. For the C programming language, shall not produce any output dependent on any
1197 behavior described in the ISO/IEC 9899:1999 standard as *unspecified*, *undefined*, or
1198 *implementation-defined*, unless the System Interfaces volume of IEEE Std 1003.1-2001
1199 specifies the behavior
- 1200 7. For the C programming language, shall not exceed any minimum implementation limit
1201 defined in the ISO/IEC 9899:1999 standard, unless the System Interfaces volume of
1202 IEEE Std 1003.1-2001 specifies a higher minimum implementation limit
- 1203 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200112L before
1204 any header is included

1205 Within IEEE Std 1003.1-2001, any restrictions placed upon a Conforming POSIX Application
1206 shall restrict a Strictly Conforming POSIX Application.

1207 2.2.2 Conforming POSIX Application

1208 2.2.2.1 ISO/IEC Conforming POSIX Application

1209 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities
1210 described in IEEE Std 1003.1-2001 and approved Conforming Language bindings for any ISO or
1211 IEC standard. Such an application shall include a statement of conformance that documents all
1212 options and limit dependencies, and all other ISO or IEC standards used.

1213 2.2.2.2 <National Body> Conforming POSIX Application

1214 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming
1215 POSIX Application in that it also may use specific standards of a single ISO/IEC member body
1216 referred to here as <National Body>. Such an application shall include a statement of
1217 conformance that documents all options and limit dependencies, and all other <National Body>
1218 standards used.

1219 2.2.3 Conforming POSIX Application Using Extensions

1220 A Conforming POSIX Application Using Extensions is an application that differs from a
1221 Conforming POSIX Application only in that it uses non-standard facilities that are consistent
1222 with IEEE Std 1003.1-2001. Such an application shall fully document its requirements for these
1223 extended facilities, in addition to the documentation required of a Conforming POSIX
1224 Application. A Conforming POSIX Application Using Extensions shall be either an ISO/IEC
1225 Conforming POSIX Application Using Extensions or a <National Body> Conforming POSIX
1226 Application Using Extensions (see Section 2.2.2.1 and Section 2.2.2.2).

1227 2.2.4 Strictly Conforming XSI Application

1228 A Strictly Conforming XSI Application is an application that requires only the facilities described
1229 in IEEE Std 1003.1-2001. Such an application:

- 1230 1. Shall accept any implementation behavior that results from actions it takes in areas
1231 described in IEEE Std 1003.1-2001 as *implementation-defined* or *unspecified*, or where
1232 IEEE Std 1003.1-2001 indicates that implementations may vary
- 1233 2. Shall not perform any actions that are described as producing *undefined* results
- 1234 3. For symbolic constants, shall accept any value in the range permitted by
1235 IEEE Std 1003.1-2001, but shall not rely on any value in the range being greater than the
1236 minimums listed or being less than the maximums listed in IEEE Std 1003.1-2001
- 1237 4. Shall not use facilities designated as *obsolescent*
- 1238 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
1239 facilities whose availability is indicated by Section 2.1.4 (on page 21)
- 1240 6. For the C programming language, shall not produce any output dependent on any
1241 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*
1242 *defined*, unless the System Interfaces volume of IEEE Std 1003.1-2001 specifies the behavior
- 1243 7. For the C programming language, shall not exceed any minimum implementation limit
1244 defined in the ISO C standard, unless the System Interfaces volume of
1245 IEEE Std 1003.1-2001 specifies a higher minimum implementation limit
- 1246 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 600 before any
1247 header is included

1248 Within IEEE Std 1003.1-2001, any restrictions placed upon a Conforming POSIX Application
1249 shall restrict a Strictly Conforming XSI Application.

1250 **2.2.5 Conforming XSI Application Using Extensions**

1251 A Conforming XSI Application Using Extensions is an application that differs from a Strictly
1252 Conforming XSI Application only in that it uses non-standard facilities that are consistent with
1253 IEEE Std 1003.1-2001. Such an application shall fully document its requirements for these
1254 extended facilities, in addition to the documentation required of a Strictly Conforming XSI
1255 Application.

1256 **2.3 Language-Dependent Services for the C Programming Language**

1257 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX
1258 conformance as described in Section 2.1.3 (on page 18).

1259 **2.4 Other Language-Related Specifications**

1260 IEEE Std 1003.1-2001 is currently specified in terms of the shell command language and ISO C.
1261 Bindings to other programming languages are being developed.

1262 If conformance to IEEE Std 1003.1-2001 is claimed for implementation of any programming
1263 language, the implementation of that language shall support the use of external symbols distinct
1264 to at least 31 bytes in length in the source program text. (That is, identifiers that differ at or
1265 before the thirty-first byte shall be distinct.) If a national or international standard governing a
1266 language defines a maximum length that is less than this value, the language-defined maximum
1267 shall be supported. External symbols that differ only by case shall be distinct when the character
1268 set in use distinguishes uppercase and lowercase characters and the language permits (or
1269 requires) uppercase and lowercase characters to be distinct in external symbols.

Definitions

1271

1272 For the purposes of IEEE Std 1003.1-2001, the terms and definitions given in Chapter 3 apply.

1273 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and
1274 definitions given in this chapter are used elsewhere in text related to extensions and options,
1275 they are shaded as appropriate.

1276 3.1 Abortive Release

1277 An abrupt termination of a network connection that may result in the loss of data.

1278 3.2 Absolute Pathname

1279 A pathname beginning with a single or more than two slashes; see also Section 3.266 (on page
1280 72).

1281 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 102).

1282 3.3 Access Mode

1283 A particular form of access permitted to a file.

1284 3.4 Additional File Access Control Mechanism

1285 An implementation-defined mechanism that is layered upon the access control mechanisms
1286 defined here, but which do not grant permissions beyond those defined herein, although they
1287 may further restrict them.

1288 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 99).

1289 3.5 Address Space

1290 The memory locations that can be referenced by a process or the threads of a process.

1291 3.6 Advisory Information

1292 An interface that advises the implementation on (portable) application behavior so that it can
1293 optimize the system.

1294 3.7 Affirmative Response

1295 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
1296 keyword **yesexpr**, matching an extended regular expression in the current locale.

1297 **Note:** The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 152).

1298 3.8 Alert

1299 To cause the user's terminal to give some audible or visual indication that an error or some other
1300 event has occurred. When the standard output is directed to a terminal device, the method for
1301 alerting the terminal user is unspecified. When the standard output is not directed to a terminal
1302 device, the alert is accomplished by writing the <alert> to standard output (unless the utility
1303 description indicates that the use of standard output produces undefined results in this case).

1304 3.9 Alert Character (<alert>)

1305 A character that in the output stream should cause a terminal to alert its user via a visual or
1306 audible notification. It is the character designated by '`\a`' in the C language. It is unspecified
1307 whether this character is the exact sequence transmitted to an output device by the system to
1308 accomplish the alert function.

1309 3.10 Alias Name

1310 In the shell command language, a word consisting solely of underscores, digits, and alphabets
1311 from the portable character set and any of the following characters: '`!`', '`%`', '`'`', '`,`', '`@`'.

1312 Implementations may allow other characters within alias names as an extension.

1313 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 115).

1314 3.11 Alignment

1315 A requirement that objects of a particular type be located on storage boundaries with addresses
1316 that are particular multiples of a byte address.

1317 **Note:** See also the ISO C standard, Section B3.

1318 3.12 Alternate File Access Control Mechanism

1319 An implementation-defined mechanism that is independent of the access control mechanisms
1320 defined herein, and which if enabled on a file may either restrict or extend the permissions of a
1321 given user. IEEE Std 1003.1-2001 defines when such mechanisms can be enabled and when they
1322 are disabled.

1323 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 99).

1324 **3.13 Alternate Signal Stack**

1325 Memory associated with a thread, established upon request by the implementation for a thread,
1326 separate from the thread signal stack, in which signal handlers responding to signals sent to that
1327 thread may be executed.

1328 **3.14 Ancillary Data**

1329 Protocol-specific, local system-specific, or optional information. The information can be both
1330 local or end-to-end significant, header information, part of a data portion, protocol-specific, and
1331 implementation or system-specific.

1332 **3.15 Angle Brackets**

1333 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase
1334 “enclosed in angle brackets”, the symbol '`<`' immediately precedes the object to be enclosed,
1335 and '`>`' immediately follows it. When describing these characters in the portable character set,
1336 the names `<less-than-sign>` and `<greater-than-sign>` are used.

1337 **3.16 Application**

1338 A computer program that performs some desired function.

1339 **3.17 Application Address**

1340 Endpoint address of a specific application.

1341 **3.18 Application Program Interface (API)**

1342 The definition of syntax and semantics for providing computer system services.

1343 **3.19 Appropriate Privileges**

1344 An implementation-defined means of associating privileges with a process with regard to the
1345 function calls, function call options, and the commands that need special privileges. There may
1346 be zero or more such means. These means (or lack thereof) are described in the conformance
1347 document.

1348 **Note:** Function calls are defined in the System Interfaces volume of IEEE Std 1003.1-2001, and
1349 commands are defined in the Shell and Utilities volume of IEEE Std 1003.1-2001.

1350 **3.20 Argument**

1351 In the shell command language, a parameter passed to a utility as the equivalent of a single
1352 string in the *argv* array created by one of the *exec* functions. An argument is one of the options,
1353 option-arguments, or operands following the command name.

1354 **Note:** The Utility Argument Syntax is defined in detail in Section 12.1 (on page 201) and the Shell and
1355 Utilities volume of IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution.

1356 In the C language, an expression in a function call expression or a sequence of preprocessing
1357 tokens in a function-like macro invocation.

1358 **3.21 Arm (a Timer)**

1359 To start a timer measuring the passage of time, enabling notifying a process when the specified
1360 time or time interval has passed.

1361 **3.22 Asterisk**

1362 The character ' * '.

1363 **3.23 Async-Cancel-Safe Function**

1364 A function that may be safely invoked by an application while the asynchronous form of
1365 cancellation is enabled. No function is async-cancel-safe unless explicitly described as such. |

1366 **3.24 Asynchronous Events**

1367 Events that occur independently of the execution of the application.

1368 **3.25 Asynchronous Input and Output**

1369 A functionality enhancement to allow an application process to queue data input and output
1370 commands with asynchronous notification of completion.

1371 **3.26 Async-Signal-Safe Function**

1372 A function that may be invoked, without restriction, from signal-catching functions. No function
1373 is async-signal-safe unless explicitly described as such.

1374 **3.27 Asynchronously-Generated Signal**

1375 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals
1376 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a
1377 property of how the signal was generated and not a property of the signal number. All signals
1378 may be generated asynchronously.

1379 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of IEEE Std 1003.1-2001.

1380 **3.28 Asynchronous I/O Completion**

1381 For an asynchronous read or write operation, when a corresponding synchronous read or write
1382 would have completed and when any associated status fields have been updated.

1383 **3.29 Asynchronous I/O Operation**

1384 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from
1385 further use of the processor.

1386 This implies that the process and the I/O operation may be running concurrently.

1387 **3.30 Authentication**

1388 The process of validating a user or process to verify that the user or process is not a counterfeit.

1389 **3.31 Authorization**

1390 The process of verifying that a user or process has permission to use a resource in the manner
1391 requested.

1392 To ensure security, the user or process would also need to be authenticated before granting
1393 access.

1394 **3.32 Background Job**

1395 See *Background Process Group* in Section 3.34.

1396 **3.33 Background Process**

1397 A process that is a member of a background process group.

1398 **3.34 Background Process Group (or Background Job)**

1399 Any process group, other than a foreground process group, that is a member of a session that
1400 has established a connection with a controlling terminal.

1401 **3.35 Backquote**

1402 The character ' ` ', also known as a grave accent.

1403 **3.36 Backslash**

1404 The character ' \ ', also known as a reverse solidus.

1405 **3.37 Backspace Character (<backspace>)**

1406 A character that, in the output stream, should cause printing (or displaying) to occur one column
1407 position previous to the position about to be printed. If the position about to be printed is at the
1408 beginning of the current line, the behavior is unspecified. It is the character designated by ' \b '
1409 in the C language. It is unspecified whether this character is the exact sequence transmitted to an
1410 output device by the system to accomplish the backspace function. The <backspace> defined
1411 here is not necessarily the ERASE special character.

1412 **Note:** Special Characters are defined in detail in Section 11.1.9 (on page 191).

1413 **3.38 Barrier**

1414 A synchronization object that allows multiple threads to synchronize at a particular point in
1415 their execution.

1416 **3.39 Base Character**

1417 One of the set of characters defined in the Latin alphabet. In Western European languages other
1418 than English, these characters are commonly used with diacritical marks (accents, cedilla, and so
1419 on) to extend the range of characters in an alphabet.

1420 **3.40 Basename**

1421 The final, or only, filename in a pathname.

1422 **3.41 Basic Regular Expression (BRE)**

1423 A regular expression (see Section 3.316 (on page 79)) used by the majority of utilities that select
1424 strings from a set of character strings.

1425 **Note:** Basic Regular Expressions are described in detail in Section 9.3 (on page 171).

1426 **3.42 Batch Access List**

1427 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a
1428 batch queue.

1429 A batch access list is associated with a batch queue. A batch server uses the batch access list of a
1430 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

1431 **3.43 Batch Administrator**

1432 A user that is authorized to modify all the attributes of queues and jobs and to change the status
1433 of a batch server.

1434 **3.44 Batch Client**

1435 A computational entity that utilizes batch services by making requests of batch servers.
1436 Batch clients often provide the means by which users access batch services, although a batch
1437 server may act as a batch client by virtue of making requests of another batch server.

1438 **3.45 Batch Destination**

1439 The batch server in a batch system to which a batch job should be sent for processing.
1440 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.
1441 A batch destination may consist of a batch server-specific portion, a network-wide portion, or
1442 both. The batch server-specific portion is referred to as the “batch queue”. The network-wide
1443 portion is referred to as a “batch server name”.

1444 **3.46 Batch Destination Identifier**

1445 A string that identifies a specific batch destination.
1446 A string of characters in the portable character set used to specify a particular batch destination.
1447 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 115).

1448 **3.47 Batch Directive**

1449 A line from a file that is interpreted by the batch server. The line is usually in the form of a
1450 comment and is an additional means of passing options to the *qsub* utility.
1451 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

1452 **3.48 Batch Job**

1453 A set of computational tasks for a computing system.
1454 Batch jobs are managed by batch servers.
1455 Once created, a batch job may be executing or pending execution. A batch job that is executing
1456 has an associated session leader (a process) that initiates and monitors the computational tasks
1457 of the batch job.

1458 **3.49 Batch Job Attribute**

1459 A named data type whose value affects the processing of a batch job.

1460 The values of the attributes of a batch job affect the processing of that job by the batch server
1461 that manages the batch job.

1462 **3.50 Batch Job Identifier**

1463 A unique name for a batch job. A name that is unique among all other batch job identifiers in a
1464 batch system and that identifies the batch server to which the batch job was originally
1465 submitted.

1466 **3.51 Batch Job Name**

1467 A label that is an attribute of a batch job. The batch job name is not necessarily unique.

1468 **3.52 Batch Job Owner**

1469 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see
1470 also Section 3.426 (on page 94)) and *hostname* is a network host name.

1471 **3.53 Batch Job Priority**

1472 A value specified by the user that may be used by an implementation to determine the order in
1473 which batch jobs are selected to be executed. Job priority has a numeric value in the range –1 024
1474 to 1 023.

1475 **Note:** The batch job priority is not the execution priority (nice value) of the batch job.

1476 **3.54 Batch Job State**

1477 An attribute of a batch job which determines the types of requests that the batch server that
1478 manages the batch job can accept for the batch job. Valid states include QUEUED, RUNNING,
1479 HELD, WAITING, EXITING, and TRANSITING.

1480 **3.55 Batch Name Service**

1481 A service that assigns batch names that are unique within the batch name space, and that can
1482 translate a unique batch name into the location of the named batch entity.

1483 **3.56 Batch Name Space**

1484 The environment within which a batch name is known to be unique.

1485 **3.57 Batch Node**

1486 A host containing part or all of a batch system.

1487 A batch node is a host meeting at least one of the following conditions:

- 1488 • Capable of executing a batch client
- 1489 • Contains a routing batch queue
- 1490 • Contains an execution batch queue

1491 **3.58 Batch Operator**

1492 A user that is authorized to modify some, but not all, of the attributes of jobs and queues, and
1493 may change the status of the batch server.

1494 **3.59 Batch Queue**

1495 A manageable object that represents a set of batch jobs and is managed by a single batch server.

1496 **Note:** A set of batch jobs is called a batch queue largely for historical reasons. Jobs are selected from
1497 the batch queue for execution based on attributes such as priority, resource requirements, and
1498 hold conditions.

1499 See also the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 3.1.2, Batch Queues.

1500 **3.60 Batch Queue Attribute**

1501 A named data type whose value affects the processing of all batch jobs that are members of the
1502 batch queue.

1503 A batch queue has attributes that affect the processing of batch jobs that are members of the
1504 batch queue.

1505 **3.61 Batch Queue Position**

1506 The place, relative to other jobs in the batch queue, occupied by a particular job in a batch queue.
1507 This is defined in part by submission time and priority; see also Section 3.62.

1508 **3.62 Batch Queue Priority**

1509 The maximum job priority allowed for any batch job in a given batch queue.

1510 The batch queue priority is set and may be changed by users with appropriate privilege. The
1511 priority is bounded in an implementation-defined manner.

1512 **3.63 Batch Rerunability**

1513 An attribute of a batch job indicating that it may be rerun after an abnormal termination from
1514 the beginning without affecting the validity of the results.

1515 3.64 Batch Restart

1516 The action of resuming the processing of a batch job from the point of the last checkpoint.
1517 Typically, this is done if the batch job has been interrupted because of a system failure.

1518 3.65 Batch Server

1519 A computational entity that provides batch services.

1520 3.66 Batch Server Name

1521 A string of characters in the portable character set used to specify a particular server in a
1522 network.

1523 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 115).

1524 3.67 Batch Service

1525 Computational and organizational services performed by a batch system on behalf of batch jobs.

1526 Batch services are of two types: requested and deferred.

1527 **Note:** Batch Services are listed in the Shell and Utilities volume of IEEE Std 1003.1-2001, Table 3-5,
1528 Batch Services Summary.

1529 3.68 Batch Service Request

1530 A solicitation of services from a batch client to a batch server.

1531 A batch service request may entail the exchange of any number of messages between the batch
1532 client and the batch server.

1533 When naming specific types of service requests, the term “request” is qualified by the type of
1534 request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

1535 3.69 Batch Submission

1536 The process by which a batch client requests that a batch server create a batch job via a *Queue Job*
1537 *Request* to perform a specified computational task.

1538 3.70 Batch System

1539 A collection of one or more batch servers.

1540 **3.71 Batch Target User**

1541 The name of a user on the batch destination batch server.

1542 The target user is the user name under whose account the batch job is to execute on the
1543 destination batch server.

1544 **3.72 Batch User**

1545 A user who is authorized to make use of batch services.

1546 **3.73 Bind**

1547 The process of assigning a network address to an endpoint.

1548 **3.74 Blank Character (<blank>)**

1549 One of the characters that belong to the **blank** character class as defined via the *LC_CTYPE*
1550 category in the current locale. In the POSIX locale, a <blank> is either a <tab> or a <space>.

1551 **3.75 Blank Line**

1552 A line consisting solely of zero or more <blank>s terminated by a <newline>; see also Section
1553 3.144 (on page 55).

1554 **3.76 Blocked Process (or Thread)**

1555 A process (or thread) that is waiting for some condition (other than the availability of a
1556 processor) to be satisfied before it can continue execution.

1557 **3.77 Blocking**

1558 A property of an open file description that causes function calls associated with it to wait for the
1559 requested action to be performed before returning.

1560 **3.78 Block-Mode Terminal**

1561 A terminal device operating in a mode incapable of the character-at-a-time input and output
1562 operations described by some of the standard utilities.

1563 **Note:** Output Devices and Terminal Types are defined in detail in Section 10.2 (on page 185).

1564 3.79 Block Special File

1565 A file that refers to a device. A block special file is normally distinguished from a character
 1566 special file by providing access to the device in a manner such that the hardware characteristics
 1567 of the device are not visible.

1568 3.80 Braces

1569 The characters ' { ' (left brace) and ' } ' (right brace), also known as curly braces. When used in
 1570 the phrase “enclosed in (curly) braces” the symbol ' { ' immediately precedes the object to be
 1571 enclosed, and ' } ' immediately follows it. When describing these characters in the portable
 1572 character set, the names <left-brace> and <right-brace> are used.

1573 3.81 Brackets

1574 The characters ' [' (left-bracket) and '] ' (right-bracket), also known as square brackets. When
 1575 used in the phrase “enclosed in (square) brackets” the symbol ' [' immediately precedes the
 1576 object to be enclosed, and '] ' immediately follows it. When describing these characters in the
 1577 portable character set, the names <left-square-bracket> and <right-square-bracket> are used.

1578 3.82 Broadcast

1579 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and
 1580 RFC 922.

1581 3.83 Built-In Utility (or Built-In)

1582 A utility implemented within a shell. The utilities referred to as special built-ins have special
 1583 qualities. Unless qualified, the term “built-in” includes the special built-in utilities. Regular
 1584 built-ins are not required to be actually built into the shell on the implementation, but they do
 1585 have special command-search qualities.

1586 **Note:** Special Built-In Utilities are defined in detail in the Shell and Utilities volume of
 1587 IEEE Std 1003.1-2001, Section 2.14, Special Built-In Utilities.

1588 Regular Built-In Utilities are defined in detail in the Shell and Utilities volume of
 1589 IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution.

1590 3.84 Byte

1591 An individually addressable unit of data storage that is exactly an octet, used to store a character
 1592 or a portion of a character; see also Section 3.87 (on page 47). A byte is composed of a
 1593 contiguous sequence of 8 bits. The least significant bit is called the “low-order” bit; the most
 1594 significant is called the “high-order” bit.

1595 **Note:** The definition of byte from the ISO C standard is broader than the above and might
 1596 accommodate hardware architectures with different sized addressable units than octets.

1597 3.85 Byte Input/Output Functions

1598 The functions that perform byte-oriented input from streams or byte-oriented output to streams:
 1599 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *printf()*,
 1600 *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1601 **Note:** Functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-2001.

1602 3.86 Carriage-Return Character (<carriage-return>)

1603 A character that in the output stream indicates that printing should start at the beginning of the
 1604 same physical line in which the <carriage-return> occurred. It is the character designated by
 1605 ‘\r’ in the C language. It is unspecified whether this character is the exact sequence
 1606 transmitted to an output device by the system to accomplish the movement to the beginning of
 1607 the line.

1608 3.87 Character

1609 A sequence of one or more bytes representing a single graphic symbol or control code.

1610 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a single-byte
 1611 character is a special case of a multi-byte character. Unlike the usage in the ISO C standard,
 1612 *character* here has no necessary relationship with storage space, and *byte* is used when storage
 1613 space is discussed.

1614 See the definition of the portable character set in Section 6.1 (on page 115) for a further
 1615 explanation of the graphical representations of (abstract) characters, as opposed to character
 1616 encodings.

1617 3.88 Character Array

1618 An array of elements of type **char**.

1619 3.89 Character Class

1620 A named set of characters sharing an attribute associated with the name of the class. The classes
 1621 and the characters that they contain are dependent on the value of the *LC_CTYPE* category in the
 1622 current locale.

1623 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 126).

1624 3.90 Character Set

1625 A finite set of different characters used for the representation, organization, or control of data.

1626 **3.91 Character Special File**

1627 A file that refers to a device. One specific type of character special file is a terminal device file.

1628 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 187).

1629 **3.92 Character String**

1630 A contiguous sequence of characters terminated by and including the first null byte.

1631 **3.93 Child Process**

1632 A new process created (by *fork()*, *posix_spawn()*, or *posix_spawnnp()*) by a given process. A child
1633 process remains the child of the creating process as long as both processes continue to exist.

1634 **Note:** The *fork()*, *posix_spawn()*, and *posix_spawnnp()* functions are defined in detail in the System
1635 Interfaces volume of IEEE Std 1003.1-2001.

1636 **3.94 Circumflex**

1637 The character '^'.

1638 **3.95 Clock**

1639 A software or hardware object that can be used to measure the apparent or actual passage of
1640 time.

1641 The current value of the time measured by a clock can be queried and, possibly, set to a value
1642 within the legal range of the clock.

1643 **3.96 Clock Jump**

1644 The difference between two successive distinct values of a clock, as observed from the
1645 application via one of the “get time” operations.

1646 **3.97 Clock Tick**

1647 An interval of time; an implementation-defined number of these occur each second. Clock ticks
1648 are one of the units that may be used to express a value found in type **clock_t**.

1649 **3.98 Coded Character Set**

1650 A set of unambiguous rules that establishes a character set and the one-to-one relationship
1651 between each character of the set and its bit representation.

1652 3.99 Codeset

1653 The result of applying rules that map a numeric code value to each element of a character set. An
1654 element of a character set may be related to more than one numeric code value but the reverse is
1655 not true. However, for state-dependent encodings the relationship between numeric code values
1656 and elements of a character set may be further controlled by state information. The character set
1657 may contain fewer elements than the total number of possible numeric code values; that is, some
1658 code values may be unassigned.

1659 **Note:** Character Encoding is defined in detail in Section 6.2 (on page 118).

1660 3.100 Collating Element

1661 The smallest entity used to determine the logical ordering of character or wide-character strings;
1662 see also Section 3.102. A collating element consists of either a single character, or two or more
1663 characters collating as a single entity. The value of the *LC_COLLATE* category in the current
1664 locale determines the current set of collating elements.

1665 3.101 Collation

1666 The logical ordering of character or wide-character strings according to defined precedence
1667 rules. These rules identify a collation sequence between the collating elements, and such
1668 additional rules that can be used to order strings consisting of multiple collating elements.

1669 3.102 Collation Sequence

1670 The relative order of collating elements as determined by the setting of the *LC_COLLATE*
1671 category in the current locale. The collation sequence is used for sorting and is determined from
1672 the collating weights assigned to each collating element. In the absence of weights, the collation
1673 sequence is the order in which collating elements are specified between **order_start** and
1674 **order_end** keywords in the *LC_COLLATE* category.

1675 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to
1676 the limit {*COLL_WEIGHTS_MAX*}. On each level, elements may be given the same weight (at
1677 the primary level, called an equivalence class; see also Section 3.150 (on page 55)) or be omitted
1678 from the sequence. Strings that collate equally using the first assigned weight (primary ordering)
1679 are then compared using the next assigned weight (secondary ordering), and so on.

1680 **Note:** {*COLL_WEIGHTS_MAX*} is defined in detail in **<limits.h>**.

1681 3.103 Column Position

1682 A unit of horizontal measure related to characters in a line.

1683 It is assumed that each character in a character set has an intrinsic column width independent of
1684 any output device. Each printable character in the portable character set has a column width of
1685 one. The standard utilities, when used as described in IEEE Std 1003.1-2001, assume that all
1686 characters have integral column widths. The column width of a character is not necessarily
1687 related to the internal representation of the character (numbers of bits or bytes).

1688 The column position of a character in a line is defined as one plus the sum of the column widths
1689 of the preceding characters in the line. Column positions are numbered starting from 1.

1690 **3.104 Command**

1691 A directive to the shell to perform a particular task.

1692 **Note:** Shell Commands are defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001,
1693 Section 2.9, Shell Commands.

1694 **3.105 Command Language Interpreter**

1695 An interface that interprets sequences of text input as commands. It may operate on an input
1696 stream or it may interactively prompt and read commands from a terminal. It is possible for
1697 applications to invoke utilities through a number of interfaces, which are collectively considered
1698 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*
1699 function, although *popen()* and the various forms of *exec* may also be considered to behave as
1700 interpreters.

1701 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

1702 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces volume
1703 of IEEE Std 1003.1-2001.

1704 **3.106 Composite Graphic Symbol**

1705 A graphic symbol consisting of a combination of two or more other graphic symbols in a single
1706 character position, such as a diacritical mark and a base character.

1707 **3.107 Condition Variable**

1708 A synchronization object which allows a thread to suspend execution, repeatedly, until some
1709 associated predicate becomes true. A thread whose execution is suspended on a condition
1710 variable is said to be blocked on the condition variable.

1711 **3.108 Connection**

1712 An association established between two or more endpoints for the transfer of data

1713 **3.109 Connection Mode**

1714 The transfer of data in the context of a connection; see also Section 3.110.

1715 **3.110 Connectionless Mode**

1716 The transfer of data other than in the context of a connection; see also Section 3.109 and Section
1717 3.123 (on page 52).

1718 **3.111 Control Character**

1719 A character, other than a graphic character, that affects the recording, processing, transmission,
1720 or interpretation of text.

1721 **3.112 Control Operator**

1722 In the shell command language, a token that performs a control function. It is one of the
1723 following symbols:

1724 & && () ; ;; newline | ||

1725 The end-of-input indicator used internally by the shell is also considered a control operator.

1726 **Note:** Token Recognition is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001,
1727 Section 2.3, Token Recognition.

1728 **3.113 Controlling Process**

1729 The session leader that established the connection to the controlling terminal. If the terminal
1730 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be
1731 the controlling process.

1732 **3.114 Controlling Terminal**

1733 A terminal that is associated with a session. Each session may have at most one controlling
1734 terminal associated with it, and a controlling terminal is associated with exactly one session.
1735 Certain input sequences from the controlling terminal cause signals to be sent to all processes in
1736 the process group associated with the controlling terminal.

1737 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 187).

1738 **3.115 Conversion Descriptor**

1739 A per-process unique value used to identify an open codeset conversion.

1740 **3.116 Core File**

1741 A file of unspecified format that may be generated when a process terminates abnormally.

1742 **3.117 CPU Time (Execution Time)**

1743 The time spent executing a process or thread, including the time spent executing system services
1744 on behalf of that process or thread. If the Threads option is supported, then the value of the
1745 CPU-time clock for a process is implementation-defined. With this definition the sum of all the
1746 execution times of all the threads in a process might not equal the process execution time, even
1747 in a single-threaded process, because implementations may differ in how they account for time
1748 during context switches or for other reasons.

1749 3.118 CPU-Time Clock

1750 A clock that measures the execution time of a particular process or thread.

1751 3.119 CPU-Time Timer

1752 A timer attached to a CPU-time clock.

1753 3.120 Current Job

1754 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There
1755 is at most one current job; see also Section 3.203 (on page 63).

1756 3.121 Current Working Directory

1757 See *Working Directory* in Section 3.436 (on page 96).

1758 3.122 Cursor Position

1759 The line and column position on the screen denoted by the terminal's cursor.

1760 3.123 Datagram

1761 A unit of data transferred from one endpoint to another in connectionless mode service.

1762 3.124 Data Segment

1763 Memory associated with a process, that can contain dynamically allocated data.

1764 3.125 Deferred Batch Service

1765 A service that is performed as a result of events that are asynchronous with respect to requests.

1766 **Note:** Once a batch job has been created, it is subject to deferred services.

1767 3.126 Device

1768 A computer peripheral or an object that appears to the application as such.

1769 3.127 Device ID

1770 A non-negative integer used to identify a device.

1771 3.128 Directory

1772 A file that contains directory entries. No two directory entries in the same directory have the
1773 same name.

1774 3.129 Directory Entry (or Link)

1775 An object that associates a filename with a file. Several directory entries can associate names
1776 with the same file.

1777 3.130 Directory Stream

1778 A sequence of all the directory entries in a particular directory. An open directory stream may be
1779 implemented using a file descriptor.

1780 3.131 Disarm (a Timer)

1781 To stop a timer from measuring the passage of time, disabling any future process notifications
1782 (until the timer is armed again).

1783 3.132 Display

1784 To output to the user's terminal. If the output is not directed to a terminal, the results are
1785 undefined.

1786 3.133 Display Line

1787 A line of text on a physical device or an emulation thereof. Such a line will have a maximum
1788 number of characters which can be presented.

1789 **Note:** This may also be written as “line on the display”.

1790 3.134 Dollar Sign

1791 The character ' \$ '.

1792 3.135 Dot

1793 In the context of naming files, the filename consisting of a single dot character (' . ').

1794 **Note:** In the context of shell special built-in utilities, see *dot* in the Shell and Utilities volume of
1795 IEEE Std 1003.1-2001, Section 2.14, Special Built-In Utilities.

1796 Pathname Resolution is defined in detail in Section 4.11 (on page 102).

1797 3.136 Dot-Dot

1798 The filename consisting solely of two dot characters (" . . ").

1799 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 102).

1800 3.137 Double-Quote

1801 The character ' " ', also known as quotation-mark.

1802 **Note:** The “double” adjective in this term refers to the two strokes in the character glyph.
1803 IEEE Std 1003.1-2001 never uses the term “double-quote” to refer to two apostrophes or
1804 quotation marks.

1805 3.138 Downshifting

1806 The conversion of an uppercase character that has a single-character lowercase representation
1807 into this lowercase representation.

1808 3.139 Driver

1809 A module that controls data transferred to and received from devices.

1810 **Note:** Drivers are traditionally written to be a part of the system implementation, although they are
1811 frequently written separately from the writing of the implementation. A driver may contain
1812 processor-specific code, and therefore be non-portable.

1813 3.140 Effective Group ID

1814 An attribute of a process that is used in determining various permissions, including file access
1815 permissions; see also Section 3.188 (on page 61).

1816 3.141 Effective User ID

1817 An attribute of a process that is used in determining various permissions, including file access
1818 permissions; see also Section 3.425 (on page 94).

1819 3.142 Eight-Bit Transparency

1820 The ability of a software component to process 8-bit characters without modifying or utilizing
1821 any part of the character in a way that is inconsistent with the rules of the current coded
1822 character set.

1823 3.143 Empty Directory

1824 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link
1825 to it, in dot-dot. No other links to the directory may exist. It is unspecified whether an
1826 implementation can ever consider the root directory to be empty.

1827 3.144 Empty Line

1828 A line consisting of only a <newline>; see also Section 3.75 (on page 45).

1829 3.145 Empty String (or Null String)

1830 A string whose first byte is a null byte.

1831 3.146 Empty Wide-Character String

1832 A wide-character string whose first element is a null wide-character code.

1833 3.147 Encoding Rule

1834 The rules used to convert between wide-character codes and multi-byte character codes.

1835 **Note:** Stream Orientation and Encoding Rules are defined in detail in the System Interfaces volume
1836 of IEEE Std 1003.1-2001, Section 2.5.2, Stream Orientation and Encoding Rules.

1837 3.148 Entire Regular Expression

1838 The concatenated set of one or more basic regular expressions or extended regular expressions
1839 that make up the pattern specified for string selection.

1840 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 169).

1841 3.149 Epoch

1842 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time
1843 (UTC).

1844 **Note:** See also Seconds Since the Epoch defined in Section 4.14 (on page 104).

1845 3.150 Equivalence Class

1846 A set of collating elements with the same primary collation weight.

1847 Elements in an equivalence class are typically elements that naturally group together, such as all
1848 accented letters based on the same base letter.

1849 The collation order of elements within an equivalence class is determined by the weights
1850 assigned on any subsequent levels after the primary weight.

1851 3.151 Era

1852 A locale-specific method for counting and displaying years.

1853 **Note:** The *LC_TIME* category is defined in detail in Section 7.3.5 (on page 147).

1854 **3.152 Event Management**

1855 The mechanism that enables applications to register for and be made aware of external events
1856 such as data becoming available for reading.

1857 **3.153 Executable File**

1858 A regular file acceptable as a new process image file by the equivalent of the *exec* family of
1859 functions, and thus usable as one form of a utility. The standard utilities described as compilers
1860 can produce executable files, but other unspecified methods of producing executable files may
1861 also be provided. The internal format of an executable file is unspecified, but a conforming
1862 application cannot assume an executable file is a text file.

1863 **3.154 Execute**

1864 To perform command search and execution actions, as defined in the Shell and Utilities volume
1865 of IEEE Std 1003.1-2001; see also Section 3.200 (on page 62).

1866 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume of
1867 IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution.

1868 **3.155 Execution Time**

1869 See *CPU Time* in Section 3.117 (on page 51).

1870 **3.156 Execution Time Monitoring**

1871 A set of execution time monitoring primitives that allow online measuring of thread and process
1872 execution times.

1873 **3.157 Expand**

1874 In the shell command language, when not qualified, the act of applying word expansions.

1875 **Note:** Word Expansions are defined in detail in the Shell and Utilities volume of
1876 IEEE Std 1003.1-2001, Section 2.6, Word Expansions.

1877 **3.158 Extended Regular Expression (ERE)**

1878 A regular expression (see also Section 3.316 (on page 79)) that is an alternative to the Basic
1879 Regular Expression using a more extensive syntax, occasionally used by some utilities.

1880 **Note:** Extended Regular Expressions are described in detail in Section 9.4 (on page 175).

1881 **3.159 Extended Security Controls**

1882 Implementation-defined security controls allowed by the file access permission and appropriate
1883 privilege (see also Section 3.19 (on page 37)) mechanisms, through which an implementation can
1884 support different security policies from those described in IEEE Std 1003.1-2001.

1885 **Note:** See also Extended Security Controls defined in Section 4.3 (on page 99).

1886 File Access Permissions are defined in detail in Section 4.4 (on page 99).

1887 **3.160 Feature Test Macro**

1888 A macro used to determine whether a particular set of features is included from a header.

1889 **Note:** See also the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.2, The Compilation
1890 Environment.

1891 **3.161 Field**

1892 In the shell command language, a unit of text that is the result of parameter expansion,
1893 arithmetic expansion, command substitution, or field splitting. During command processing, the
1894 resulting fields are used as the command name and its arguments.

1895 **Note:** Parameter Expansion is defined in detail in the Shell and Utilities volume of
1896 IEEE Std 1003.1-2001, Section 2.6.2, Parameter Expansion.

1897 Arithmetic Expansion is defined in detail in the Shell and Utilities volume of
1898 IEEE Std 1003.1-2001, Section 2.6.4, Arithmetic Expansion.

1899 Command Substitution is defined in detail in the Shell and Utilities volume of
1900 IEEE Std 1003.1-2001, Section 2.6.3, Command Substitution.

1901 Field Splitting is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001,
1902 Section 2.6.5, Field Splitting.

1903 For further information on command processing, see the Shell and Utilities volume of
1904 IEEE Std 1003.1-2001, Section 2.9.1, Simple Commands.

1905 **3.162 FIFO Special File (or FIFO)**

1906 A type of file with the property that data written to such a file is read on a first-in-first-out basis.

1907 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of
1908 IEEE Std 1003.1-2001, *lseek()*, *open()*, *read()*, and *write()*.

1909 **3.163 File**

1910 An object that can be written to, or read from, or both. A file has certain attributes, including
1911 access permissions and type. File types include regular file, character special file, block special
1912 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported
1913 by the implementation.

1914 **3.164 File Description**

1915 See *Open File Description* in Section 3.253 (on page 70).

1916 **3.165 File Descriptor**

1917 A per-process unique, non-negative integer used to identify an open file for the purpose of file
 1918 access. The value of a file descriptor is from zero to {OPEN_MAX}. A process can have no more
 1919 than {OPEN_MAX} file descriptors open simultaneously. File descriptors may also be used to
 1920 implement message catalog descriptors and directory streams; see also Section 3.253 (on page
 1921 70).

1922 **Note:** {OPEN_MAX} is defined in detail in **<limits.h>**.

1923 **3.166 File Group Class**

1924 The property of a file indicating access permissions for a process related to the group
 1925 identification of a process. A process is in the file group class of a file if the process is not in the
 1926 file owner class and if the effective group ID or one of the supplementary group IDs of the
 1927 process matches the group ID associated with the file. Other members of the class may be
 1928 implementation-defined.

1929 **3.167 File Mode**

1930 An object containing the file mode bits and file type of a file.

1931 **Note:** File mode bits and file types are defined in detail in **<sys/stat.h>**.

1932 **3.168 File Mode Bits**

1933 A file's file permission bits: set-user-ID-on-execution bit (S_ISUID), set-group-ID-on-execution
 1934 bit (S_ISGID), and, on directories, the restricted deletion flag bit (S_ISVTX).

1935 **Note:** File Mode Bits are defined in detail in **<sys/stat.h>**.

1936 **3.169 Filename**

1937 A name consisting of 1 to {NAME_MAX} bytes used to name a file. The characters composing
 1938 the name may be selected from the set of all character values excluding the slash character and
 1939 the null byte. The filenames dot and dot-dot have special meaning. A filename is sometimes
 1940 referred to as a “pathname component”.

1941 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 102).

1942 **3.170 Filename Portability**

1943 Filenames should be constructed from the portable filename character set because the use of
 1944 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a
 1945 colon (':') in a pathname could cause ambiguity if that pathname were included in a *PATH*
 1946 definition.)

1947 3.171 File Offset

1948 The byte position in the file where the next I/O operation begins. Each open file description
1949 associated with a regular file, block special file, or directory has a file offset. A character special
1950 file that does not refer to a terminal device may have a file offset. There is no file offset specified
1951 for a pipe or FIFO.

1952 3.172 File Other Class

1953 The property of a file indicating access permissions for a process related to the user and group
1954 identification of a process. A process is in the file other class of a file if the process is not in the
1955 file owner class or file group class.

1956 3.173 File Owner Class

1957 The property of a file indicating access permissions for a process related to the user
1958 identification of a process. A process is in the file owner class of a file if the effective user ID of
1959 the process matches the user ID of the file.

1960 3.174 File Permission Bits

1961 Information about a file that is used, along with other information, to determine whether a
1962 process has read, write, or execute/search permission to a file. The bits are divided into three
1963 parts: owner, group, and other. Each part is used with the corresponding file class of processes.
1964 These bits are contained in the file mode.

1965 **Note:** File modes are defined in detail in <sys/stat.h>.

1966 File Access Permissions are defined in detail in Section 4.4 (on page 99).

1967 3.175 File Serial Number

1968 A per-file system unique identifier for a file.

1969 3.176 File System

1970 A collection of files and certain of their attributes. It provides a name space for file serial
1971 numbers referring to those files.

1972 3.177 File Type

1973 See *File* in Section 3.163 (on page 57).

1974 **3.178 Filter**

1975 A command whose operation consists of reading data from standard input or a list of input files
 1976 and writing data to standard output. Typically, its function is to perform some transformation
 1977 on the data stream.

1978 **3.179 First Open (of a File)**

1979 When a process opens a file that is not currently an open file within any process.

1980 **3.180 Flow Control**

1981 The mechanism employed by a communications provider that constrains a sending entity to
 1982 wait until the receiving entities can safely receive additional data without loss.

1983 **3.181 Foreground Job**

1984 See *Foreground Process Group* in Section 3.183.

1985 **3.182 Foreground Process**

1986 A process that is a member of a foreground process group.

1987 **3.183 Foreground Process Group (or Foreground Job)**

1988 A process group whose member processes have certain privileges, denied to processes in
 1989 background process groups, when accessing their controlling terminal. Each session that has
 1990 established a connection with a controlling terminal has at most one process group of the session
 1991 as the foreground process group of that controlling terminal.

1992 **Note:** The General Terminal Interface is defined in detail in Chapter 11.

1993 **3.184 Foreground Process Group ID**

1994 The process group ID of the foreground process group.

1995 **3.185 Form-Feed Character (<form-feed>)**

1996 A character that in the output stream indicates that printing should start on the next page of an
 1997 output device. It is the character designated by ' \f ' in the C language. If the <form-feed> is not
 1998 the first character of an output line, the result is unspecified. It is unspecified whether this
 1999 character is the exact sequence transmitted to an output device by the system to accomplish the
 2000 movement to the next page.

2001 **3.186 Graphic Character**

2002 A member of the **graph** character class of the current locale.

2003 **Note:** The **graph** character class is defined in detail in Section 7.3.1 (on page 126).

2004 **3.187 Group Database**

2005 A system database of implementation-defined format that contains at least the following
2006 information for each group ID:

- 2007 • Group name
- 2008 • Numerical group ID
- 2009 • List of users allowed in the group

2010 The list of users allowed in the group is used by the *newgrp* utility.

2011 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

2012 **3.188 Group ID**

2013 A non-negative integer, which can be contained in an object of type **gid_t**, that is used to identify
2014 a group of system users. Each system user is a member of at least one group. When the identity
2015 of a group is associated with a process, a group ID value is referred to as a real group ID, an
2016 effective group ID, one of the supplementary group IDs, or a saved set-group-ID.

2017 **3.189 Group Name**

2018 A string that is used to identify a group; see also Section 3.187. To be portable across conforming
2019 systems, the value is composed of characters from the portable filename character set. The
2020 hyphen should not be used as the first character of a portable group name.

2021 **3.190 Hard Limit**

2022 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.
2023 A non-privileged process is restricted to only lowering its hard limit.

2024 **3.191 Hard Link**

2025 The relationship between two directory entries that represent the same file; see also Section 3.129
2026 (on page 53). The result of an execution of the *ln* utility (without the *-s* option) or the *link()*
2027 function. This term is contrasted against symbolic link; see also Section 3.372 (on page 86).

2028 **3.192 Home Directory**

2029 The directory specified by the *HOME* environment variable.

2030 3.193 Host Byte Order

2031 The arrangement of bytes in any integer type when using a specific machine architecture.

2032 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian is a
2033 format for storage of binary data in which the most significant byte is placed first, with the rest
2034 in descending order. Little-endian is a format for storage or transmission of binary data in
2035 which the least significant byte is placed first, with the rest in ascending order. See also Section
2036 4.8 (on page 101).

2037 3.194 Incomplete Line

2038 A sequence of one or more non-`<newline>`s at the end of the file.

2039 3.195 Inf

2040 A value representing +infinity or a value representing –infinity that can be stored in a floating
2041 type. Not all systems support the Inf values.

2042 3.196 Instrumented Application

2043 An application that contains at least one call to the trace point function `posix_trace_event()`. Each
2044 process of an instrumented application has a mapping of trace event names to trace event type
2045 identifiers. This mapping is used by the trace stream that is created for that process.

2046 3.197 Interactive Shell

2047 A processing mode of the shell that is suitable for direct user interaction.

2048 3.198 Internationalization

2049 The provision within a computer program of the capability of making itself adaptable to the
2050 requirements of different native languages, local customs, and coded character sets.

2051 3.199 Interprocess Communication

2052 A functionality enhancement to add a high-performance, deterministic interprocess
2053 communication facility for local communication.

2054 3.200 Invoke

2055 To perform command search and execution actions, except that searching for shell functions and
2056 special built-in utilities is suppressed; see also Section 3.154 (on page 56).

2057 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume of
2058 IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution.

2059 **3.201 Job**

2060 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all
 2061 in the same process group.

2062 **Note:** See also the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.9.2, Pipelines.

2063 **3.202 Job Control**

2064 A facility that allows users selectively to stop (suspend) the execution of processes and continue
 2065 (resume) their execution at a later point. The user typically employs this facility via the
 2066 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

2067 **3.203 Job Control Job ID**

2068 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in the
 2069 following table:

2070 **Table 3-1** Job Control Job ID Formats

2071 2072	Job Control Job ID	Meaning
2073	%%	Current job.
2074	%+	Current job.
2075	%-	Previous job.
2076	% <i>n</i>	Job number <i>n</i> .
2077	% <i>string</i>	Job whose command begins with <i>string</i> .
2078	%? <i>string</i>	Job whose command contains <i>string</i> .

2079 **3.204 Last Close (of a File)**

2080 When a process closes a file, resulting in the file not being an open file within any process.

2081 **3.205 Line**

2082 A sequence of zero or more non-`<newline>`s plus a terminating `<newline>`.

2083 **3.206 Linger**

2084 A period of time before terminating a connection, to allow outstanding data to be transferred.

2085 **3.207 Link**

2086 See *Directory Entry* in Section 3.129 (on page 53).

2087 3.208 Link Count

2088 The number of directory entries that refer to a particular file.

2089 3.209 Local Customs

2090 The conventions of a geographical area or territory for such things as date, time, and currency
2091 formats.

2092 3.210 Local Interprocess Communication (Local IPC)

2093 The transfer of data between processes in the same system.

2094 3.211 Locale

2095 The definition of the subset of a user's environment that depends on language and cultural
2096 conventions.

2097 **Note:** Locales are defined in detail in Chapter 7 (on page 123).

2098 3.212 Localization

2099 The process of establishing information within a computer system specific to the operation of
2100 particular native languages, local customs, and coded character sets.

2101 3.213 Login

2102 The unspecified activity by which a user gains access to the system. Each login is associated
2103 with exactly one login name.

2104 3.214 Login Name

2105 A user name that is associated with a login.

2106 3.215 Map

2107 To create an association between a page-aligned range of the address space of a process and
2108 some memory object, such that a reference to an address in that range of the address space
2109 results in a reference to the associated memory object. The mapped memory object is not
2110 necessarily memory-resident.

2111 **3.216 Marked Message**

2112 A STREAMS message on which a certain flag is set. Marking a message gives the application
2113 protocol-specific information. An application can use *ioctl()* to determine whether a given
2114 message is marked.

2115 **Note:** The *ioctl()* function is defined in detail in the System Interfaces volume of
2116 IEEE Std 1003.1-2001.

2117 **3.217 Matched**

2118 A state applying to a sequence of zero or more characters when the characters in the sequence
2119 correspond to a sequence of characters defined by a basic regular expression or extended regular
2120 expression pattern.

2121 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 169).

2122 **3.218 Memory Mapped Files**

2123 A facility to allow applications to access files as part of the address space.

2124 **3.219 Memory Object**

2125 One of:

- 2126 • A file (see Section 3.163 (on page 57))
- 2127 • A shared memory object (see Section 3.340 (on page 82))
- 2128 • A typed memory object (see Section 3.418 (on page 93))

2129 When used in conjunction with *mmap()*, a memory object appears in the address space of the
2130 calling process.

2131 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of
2132 IEEE Std 1003.1-2001.

2133 **3.220 Memory-Resident**

2134 The process of managing the implementation in such a way as to provide an upper bound on
2135 memory access times.

2136 **3.221 Message**

2137 In the context of programmatic message passing, information that can be transferred between
2138 processes or threads by being added to and removed from a message queue. A message consists
2139 of a fixed-size message buffer.

2140 **3.222 Message Catalog**

2141 In the context of providing natural language messages to the user, a file or storage area
2142 containing program messages, command prompts, and responses to prompts for a particular
2143 native language, territory, and codeset.

2144 **3.223 Message Catalog Descriptor**

2145 In the context of providing natural language messages to the user, a per-process unique value
2146 used to identify an open message catalog. A message catalog descriptor may be implemented
2147 using a file descriptor.

2148 **3.224 Message Queue**

2149 In the context of programmatic message passing, an object to which messages can be added and
2150 removed. Messages may be removed in the order in which they were added or in priority order.

2151 **3.225 Mode**

2152 A collection of attributes that specifies a file's type and its access permissions.

2153 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 99).

2154 **3.226 Monotonic Clock**

2155 A clock whose value cannot be set via *clock_settime()* and which cannot have negative clock
2156 jumps.

2157 **3.227 Mount Point**

2158 Either the system root directory or a directory for which the *st_dev* field of structure **stat** differs
2159 from that of its parent directory.

2160 **Note:** The **stat** structure is defined in detail in `<sys/stat.h>`.

2161 **3.228 Multi-Character Collating Element**

2162 A sequence of two or more characters that collate as an entity. For example, in some coded
2163 character sets, an accented character is represented by a non-spacing accent, followed by the
2164 letter. Other examples are the Spanish elements *ch* and *ll*.

2165 **3.229 Mutex**

2166 A synchronization object used to allow multiple threads to serialize their access to shared data.
2167 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has
2168 locked a mutex becomes its owner and remains the owner until that same thread unlocks the
2169 mutex.

2170 3.230 Name

2171 In the shell command language, a word consisting solely of underscores, digits, and alphabetic
2172 from the portable character set. The first character of a name is not a digit.

2173 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 115).

2174 3.231 Named STREAM

2175 A STREAMS-based file descriptor that is attached to a name in the file system name space. All
2176 subsequent operations on the named STREAM act on the STREAM that was associated with the
2177 file descriptor until the name is disassociated from the STREAM.

2178 3.232 NaN (Not a Number)

2179 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-
2180 point numbers. Not all systems support NaN values.

2181 3.233 Native Language

2182 A computer user's spoken or written language, such as American English, British English,
2183 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

2184 3.234 Negative Response

2185 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
2186 keyword **noexpr**, matching an extended regular expression in the current locale.

2187 **Note:** The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 152).

2188 3.235 Network

2189 A collection of interconnected hosts.

2190 **Note:** The term “network” in IEEE Std 1003.1-2001 is used to refer to the network of hosts. The term
2191 “batch system” is used to refer to the network of batch servers.

2192 3.236 Network Address

2193 A network-visible identifier used to designate specific endpoints in a network. Specific
2194 endpoints on host systems have addresses, and host systems may also have addresses.

2195 **3.237 Network Byte Order**

2196 The way of representing any integer type such that, when transmitted over a network via a
2197 network endpoint, the **int** type is transmitted as an appropriate number of octets with the most
2198 significant octet first, followed by any other octets in descending order of significance.

2199 **Note:** This order is more commonly known as big-endian ordering. See also Section 4.8 (on page 101).

2200 **3.238 Newline Character (<newline>)**

2201 A character that in the output stream indicates that printing should start at the beginning of the
2202 next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this
2203 character is the exact sequence transmitted to an output device by the system to accomplish the
2204 movement to the next line.

2205 **3.239 Nice Value**

2206 A number used as advice to the system to alter process scheduling. Numerically smaller values
2207 give a process additional preference when scheduling a process to run. Numerically larger
2208 values reduce the preference and make a process less likely to run. Typically, a process with a
2209 smaller nice value runs to completion more quickly than an equivalent process with a higher
2210 nice value. The symbol {NZERO} specifies the default nice value of the system.

2211 **3.240 Non-Blocking**

2212 A property of an open file description that causes function calls involving it to return without
2213 delay when it is detected that the requested action associated with the function call cannot be
2214 completed without unknown delay.

2215 **Note:** The exact semantics are dependent on the type of file associated with the open file description.
2216 For data reads from devices such as ttys and FIFOs, this property causes the read to return
2217 immediately when no data was available. Similarly, for writes, it causes the call to return
2218 immediately when the thread would otherwise be delayed in the write operation; for example,
2219 because no space was available. For networking, it causes functions not to await protocol
2220 events (for example, acknowledgements) to occur. See also the System Interfaces volume of
2221 IEEE Std 1003.1-2001, Section 2.10.7, Socket I/O Mode.

2222 **3.241 Non-Spacing Characters**

2223 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:1994
2224 standard coded character set, which is used in combination with other characters to form
2225 composite graphic symbols.

2226 **3.242 NUL**

2227 A character with all bits set to zero.

2228 3.243 Null Byte

2229 A byte with all bits set to zero.

2230 3.244 Null Pointer

2231 The value that is obtained by converting the number 0 into a pointer; for example, **(void *) 0**. The
2232 C language guarantees that this value does not match that of any legitimate pointer, so it is used
2233 by many functions that return pointers to indicate an error.

2234 3.245 Null String

2235 See *Empty String* in Section 3.145 (on page 55).

2236 3.246 Null Wide-Character Code

2237 A wide-character code with all bits set to zero.

2238 3.247 Number Sign

2239 The character ' # ', also known as hash sign.

2240 3.248 Object File

2241 A regular file containing the output of a compiler, formatted as input to a linkage editor for
2242 linking with other object files into an executable form. The methods of linking are unspecified
2243 and may involve the dynamic linking of objects at runtime. The internal format of an object file
2244 is unspecified, but a conforming application cannot assume an object file is a text file.

2245 3.249 Octet

2246 Unit of data representation that consists of eight contiguous bits.

2247 3.250 Offset Maximum

2248 An attribute of an open file description representing the largest value that can be used as a file
2249 offset.

2250 3.251 Opaque Address

2251 An address such that the entity making use of it requires no details about its contents or format.

2252 3.252 Open File

2253 A file that is currently associated with a file descriptor.

2254 3.253 Open File Description

2255 A record of how a process or group of processes is accessing a file. Each file descriptor refers to
2256 exactly one open file description, but an open file description can be referred to by more than
2257 one file descriptor. The file offset, file status, and file access modes are attributes of an open file
2258 description.

2259 3.254 Operand

2260 An argument to a command that is generally used as an object supplying information to a utility
2261 necessary to complete its processing. Operands generally follow the options in a command line.

2262 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 201).

2263 3.255 Operator

2264 In the shell command language, either a control operator or a redirection operator.

2265 3.256 Option

2266 An argument to a command that is generally used to specify changes in the utility's default
2267 behavior.

2268 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 201).

2269 3.257 Option-Argument

2270 A parameter that follows certain options. In some cases an option-argument is included within
2271 the same argument string as the option—in most cases it is the next argument.

2272 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 201).

2273 3.258 Orientation

2274 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2275 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-2001, Section
2276 2.5.2, Stream Orientation and Encoding Rules.

2277 3.259 Orphaned Process Group

2278 A process group in which the parent of every member is either itself a member of the group or is
2279 not a member of the group's session.

2280 3.260 Page

- 2281 The granularity of process memory mapping or locking.
- 2282 Physical memory and memory objects can be mapped into the address space of a process on
- 2283 page boundaries and in integral multiples of pages. Process address space can be locked into
- 2284 memory (made memory-resident) on page boundaries and in integral multiples of pages.

2285 3.261 Page Size

- 2286 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On systems
- 2287 that have segment rather than page-based memory architectures, the term “page” means a
- 2288 segment.

2289 3.262 Parameter

- 2290 In the shell command language, an entity that stores values. There are three types of parameters:
- 2291 variables (named parameters), positional parameters, and special parameters. Parameter
- 2292 expansion is accomplished by introducing a parameter with the ‘\$’ character.
- 2293 **Note:** See also the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.5, Parameters and
- 2294 Variables.
- 2295 In the C language, an object declared as part of a function declaration or definition that acquires
- 2296 a value on entry to the function, or an identifier following the macro name in a function-like
- 2297 macro definition.

2298 3.263 Parent Directory

- 2299 When discussing a given directory, the directory that both contains a directory entry for the
- 2300 given directory and is represented by the pathname dot-dot in the given directory.
- 2301 When discussing other types of files, a directory containing a directory entry for the file under
- 2302 discussion.
- 2303 This concept does not apply to dot and dot-dot.

2304 3.264 Parent Process

- 2305 The process which created (or inherited) the process under discussion.

2306 3.265 Parent Process ID

- 2307 An attribute of a new process identifying the parent of the process. The parent process ID of a
- 2308 process is the process ID of its creator, for the lifetime of the creator. After the creator’s lifetime
- 2309 has ended, the parent process ID is the process ID of an implementation-defined system process.

2310 **3.266 Pathname**

2311 A character string that is used to identify a file. In the context of IEEE Std 1003.1-2001, a
2312 pathname consists of, at most, {PATH_MAX} bytes, including the terminating null byte. It has an
2313 optional beginning slash, followed by zero or more filenames separated by slashes. A pathname
2314 may optionally contain one or more trailing slashes. Multiple successive slashes are considered
2315 to be the same as one slash.

2316 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 102).

2317 **3.267 Pathname Component**

2318 See *Filename* in Section 3.169 (on page 58).

2319 **3.268 Path Prefix**

2320 A pathname, with an optional ending slash, that refers to a directory.

2321 **3.269 Pattern**

2322 A sequence of characters used either with regular expression notation or for pathname
2323 expansion, as a means of selecting various character strings or pathnames, respectively.

2324 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 169).

2325 See also the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.6, Pathname
2326 Expansion.

2327 The syntaxes of the two types of patterns are similar, but not identical; IEEE Std 1003.1-2001
2328 always indicates the type of pattern being referred to in the immediate context of the use of the
2329 term.

2330 **3.270 Period**

2331 The character ' . '. The term “period” is contrasted with dot (see also Section 3.135 (on page
2332 53)), which is used to describe a specific directory entry.

2333 **3.271 Permissions**

2334 Attributes of an object that determine the privilege necessary to access or manipulate the object.

2335 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 99).

2336 **3.272 Persistence**

2337 A mode for semaphores, shared memory, and message queues requiring that the object and its
2338 state (including data, if any) are preserved after the object is no longer referenced by any process.

2339 Persistence of an object does not imply that the state of the object is maintained across a system
2340 crash or a system reboot.

2341 **3.273 Pipe**

2342 An object accessed by one of the pair of file descriptors created by the *pipe()* function. Once
 2343 created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO
 2344 special file when accessed in this way. It has no name in the file hierarchy.

2345 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of
 2346 IEEE Std 1003.1-2001.

2347 **3.274 Polling**

2348 A scheduling scheme whereby the local process periodically checks until the pre-specified
 2349 events (for example, read, write) have occurred.

2350 **3.275 Portable Character Set**

2351 The collection of characters that are required to be present in all locales supported by
 2352 conforming systems.

2353 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 115).

2354 This term is contrasted against the smaller portable filename character set; see also Section 3.276.

2355 **3.276 Portable Filename Character Set**

2356 The set of characters from which portable filenames are constructed.

2357 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 2358 a b c d e f g h i j k l m n o p q r s t u v w x y z
 2359 0 1 2 3 4 5 6 7 8 9 . _ -

2360 The last three characters are the period, underscore, and hyphen characters, respectively.

2361 **3.277 Positional Parameter**

2362 In the shell command language, a parameter denoted by a single digit or one or more digits in
 2363 curly braces.

2364 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
 2365 2.5.1, Positional Parameters.

2366 **3.278 Preallocation**

2367 The reservation of resources in a system for a particular use.

2368 Preallocation does not imply that the resources are immediately allocated to that use, but merely
 2369 indicates that they are guaranteed to be available in bounded time when needed.

2370 3.279 Preempted Process (or Thread)

2371 A running thread whose execution is suspended due to another thread becoming runnable at a
2372 higher priority.

2373 3.280 Previous Job

2374 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the
2375 current job exits. There is at most one previous job; see also Section 3.203 (on page 63).

2376 3.281 Printable Character

2377 One of the characters included in the **print** character classification of the *LC_CTYPE* category in
2378 the current locale.

2379 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 126).

2380 3.282 Printable File

2381 A text file consisting only of the characters included in the **print** and **space** character
2382 classifications of the *LC_CTYPE* category and the <backspace>, all in the current locale.

2383 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 126).

2384 3.283 Priority

2385 A non-negative integer associated with processes or threads whose value is constrained to a
2386 range defined by the applicable scheduling policy. Numerically higher values represent higher
2387 priorities.

2388 3.284 Priority Band

2389 The queuing order applied to normal priority STREAMS messages. High priority STREAMS
2390 messages are not grouped by priority bands. The only differentiation made by the STREAMS
2391 mechanism is between zero and non-zero bands, but specific protocol modules may differentiate
2392 between priority bands.

2393 3.285 Priority Inversion

2394 A condition in which a thread that is not voluntarily suspended (waiting for an event or time
2395 delay) is not running while a lower priority thread is running. Such blocking of the higher
2396 priority thread is often caused by contention for a shared resource.

2397 3.286 Priority Scheduling

2398 A performance and determinism improvement facility to allow applications to determine the
2399 order in which threads that are ready to run are granted access to processor resources.

2400 3.287 Priority-Based Scheduling

2401 Scheduling in which the selection of a running thread is determined by the priorities of the
2402 runnable processes or threads.

2403 3.288 Privilege

2404 See *Appropriate Privileges* in Section 3.19 (on page 37).

2405 3.289 Process

2406 An address space with one or more threads executing within that address space, and the
2407 required system resources for those threads.

2408 **Note:** Many of the system resources defined by IEEE Std 1003.1-2001 are shared among all of the
2409 threads within a process. These include the process ID, the parent process ID, process group ID,
2410 session membership, real, effective, and saved set-user-ID, real, effective, and saved set-group-
2411 ID, supplementary group IDs, current working directory, root directory, file mode creation
2412 mask, and file descriptors.

2413 3.290 Process Group

2414 A collection of processes that permits the signaling of related processes. Each process in the
2415 system is a member of a process group that is identified by a process group ID. A newly created
2416 process joins the process group of its creator.

2417 3.291 Process Group ID

2418 The unique positive integer identifier representing a process group during its lifetime.

2419 **Note:** See also Process Group ID Reuse defined in Section 4.12 (on page 103).

2420 3.292 Process Group Leader

2421 A process whose process ID is the same as its process group ID.

2422 3.293 Process Group Lifetime

2423 A period of time that begins when a process group is created and ends when the last remaining
2424 process in the group leaves the group, due either to the end of the last process' lifetime or to the
2425 last remaining process calling the *setsid()* or *setpgid()* functions.

2426 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of
2427 IEEE Std 1003.1-2001.

2428 **3.294 Process ID**

2429 The unique positive integer identifier representing a process during its lifetime.

2430 **Note:** See also Process ID Reuse defined in Section 4.12 (on page 103).

2431 **3.295 Process Lifetime**

2432 The period of time that begins when a process is created and ends when its process ID is
 2433 returned to the system. After a process is created with a *fork()* function, it is considered active.
 2434 At least one thread of control and address space exist until it terminates. It then enters an
 2435 inactive state where certain resources may be returned to the system, although some resources,
 2436 such as the process ID, are still in use. When another process executes a *wait()*, *waitid()*, or
 2437 *waitpid()* function for an inactive process, the remaining resources are returned to the system.
 2438 The last resource to be returned to the system is the process ID. At this time, the lifetime of the
 2439 process ends.

2440 **Note:** The *fork()*, *wait()*, *waitid()*, and *waitpid()* functions are defined in detail in the System
 2441 Interfaces volume of IEEE Std 1003.1-2001.

2442 **3.296 Process Memory Locking**

2443 A performance improvement facility to bind application programs into the high-performance
 2444 random access memory of a computer system. This avoids potential latencies introduced by the
 2445 operating system in storing parts of a program that were not recently referenced on secondary
 2446 memory devices.

2447 **3.297 Process Termination**

2448 There are two kinds of process termination:

- 2449 1. Normal termination occurs by a return from *main()* or when requested with the *exit()* or
 2450 *_exit()* functions.
- 2451 2. Abnormal termination occurs when requested by the *abort()* function or when some
 2452 signals are received.

2453 **Note:** The *_exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces volume
 2454 of IEEE Std 1003.1-2001.

2455 **3.298 Process-To-Process Communication**

2456 The transfer of data between processes.

2457 **3.299 Process Virtual Time**

2458 The measurement of time in units elapsed by the system clock while a process is executing.

2459 3.300 Program

2460 A prepared sequence of instructions to the system to accomplish a defined task. The term
2461 “program” in IEEE Std 1003.1-2001 encompasses applications written in the Shell Command
2462 Language, complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level
2463 languages.

2464 3.301 Protocol

2465 A set of semantic and syntactic rules for exchanging information.

2466 3.302 Pseudo-Terminal

2467 A facility that provides an interface that is identical to the terminal subsystem. A pseudo-
2468 terminal is composed of two devices: the “master device” and a “slave device”. The slave device
2469 provides processes with an interface that is identical to the terminal interface, although there
2470 need not be hardware behind that interface. Anything written on the master device is presented
2471 to the slave as an input and anything written on the slave device is presented as an input on the
2472 master side.

2473 3.303 Radix Character

2474 The character that separates the integer part of a number from the fractional part.

2475 3.304 Read-Only File System

2476 A file system that has implementation-defined characteristics restricting modifications.

2477 **Note:** File Times Update is described in detail in Section 4.7 (on page 100).

2478 3.305 Read-Write Lock

2479 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
2480 read-only access to data while allowing only one thread to have write access at any given time.
2481 They are typically used to protect data that is read-only more frequently than it is changed.

2482 Read-write locks can be used to synchronize threads in the current process and other processes if
2483 they are allocated in memory that is writable and shared among the cooperating processes and
2484 have been initialized for this behavior.

2485 3.306 Real Group ID

2486 The attribute of a process that, at the time of process creation, identifies the group of the user
2487 who created the process; see also Section 3.188 (on page 61).

2488 **3.307 Real Time**

2489 Time measured as total units elapsed by the system clock without regard to which thread is
2490 executing.

2491 **3.308 Realtime Signal Extension**

2492 A determinism improvement facility to enable asynchronous signal notifications to an
2493 application to be queued without impacting compatibility with the existing signal functions.

2494 **3.309 Real User ID**

2495 The attribute of a process that, at the time of process creation, identifies the user who created the
2496 process; see also Section 3.425 (on page 94).

2497 **3.310 Record**

2498 A collection of related data units or words which is treated as a unit.

2499 **3.311 Redirection**

2500 In the shell command language, a method of associating files with the input or output of
2501 commands.

2502 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.7,
2503 Redirection.

2504 **3.312 Redirection Operator**

2505 In the shell command language, a token that performs a redirection function. It is one of the
2506 following symbols:

2507 < > >| << >> <& >& <<- <>

2508 **3.313 Reentrant Function**

2509 A function whose effect, when called by two or more threads, is guaranteed to be as if the
2510 threads each executed the function one after another in an undefined order, even if the actual
2511 execution is interleaved.

2512 **3.314 Referenced Shared Memory Object**

2513 A shared memory object that is open or has one or more mappings defined on it.

2514 3.315 Refresh

2515 To ensure that the information on the user's terminal screen is up-to-date.

2516 3.316 Regular Expression

2517 A pattern that selects specific strings from a set of character strings.

2518 **Note:** Regular Expressions are described in detail in Chapter 9 (on page 169).

2519 3.317 Region

2520 In the context of the address space of a process, a sequence of addresses.

2521 In the context of a file, a sequence of offsets.

2522 3.318 Regular File

2523 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the
2524 system.

2525 3.319 Relative Pathname

2526 A pathname not beginning with a slash.

2527 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 102).

2528 3.320 Relocatable File

2529 A file holding code or data suitable for linking with other object files to create an executable or a
2530 shared object file.

2531 3.321 Relocation

2532 The process of connecting symbolic references with symbolic definitions. For example, when a
2533 program calls a function, the associated call instruction transfers control to the proper
2534 destination address at execution.

2535 3.322 Requested Batch Service

2536 A service that is either rejected or performed prior to a response from the service to the
2537 requester.

2538 3.323 (Time) Resolution

2539 The minimum time interval that a clock can measure or whose passage a timer can detect.

2540 3.324 Root Directory

2541 A directory, associated with a process, that is used in pathname resolution for pathnames that
2542 begin with a slash.

2543 3.325 Runnable Process (or Thread)

2544 A thread that is capable of being a running thread, but for which no processor is available.

2545 3.326 Running Process (or Thread)

2546 A thread currently executing on a processor. On multi-processor systems there may be more
2547 than one such thread in a system at a time.

2548 3.327 Saved Resource Limits

2549 An attribute of a process that provides some flexibility in the handling of unrepresentable
2550 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2551 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces volume of
2552 IEEE Std 1003.1-2001.

2553 3.328 Saved Set-Group-ID

2554 An attribute of a process that allows some flexibility in the assignment of the effective group ID
2555 attribute, as described in the *exec* family of functions and *setgid()*.

2556 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume of
2557 IEEE Std 1003.1-2001.

2558 3.329 Saved Set-User-ID

2559 An attribute of a process that allows some flexibility in the assignment of the effective user ID
2560 attribute, as described in the *exec* family of functions and *setuid()*.

2561 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume of
2562 IEEE Std 1003.1-2001.

2563 3.330 Scheduling

2564 The application of a policy to select a runnable process or thread to become a running process or
2565 thread, or to alter one or more of the thread lists.

2566 3.331 Scheduling Allocation Domain

2567 The set of processors on which an individual thread can be scheduled at any given time.

2568 **3.332 Scheduling Contention Scope**

2569 A property of a thread that defines the set of threads against which that thread competes for
2570 resources.

2571 For example, in a scheduling decision, threads sharing scheduling contention scope compete for
2572 processor resources. In IEEE Std 1003.1-2001, a thread has scheduling contention scope of either
2573 PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS.

2574 **3.333 Scheduling Policy**

2575 A set of rules that is used to determine the order of execution of processes or threads to achieve
2576 some goal.

2577 **Note:** Scheduling Policy is defined in detail in Section 4.13 (on page 103).

2578 **3.334 Screen**

2579 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a
2580 physical display device or may occupy the entire physical area of the display device.

2581 **3.335 Scroll**

2582 To move the representation of data vertically or horizontally relative to the terminal screen.
2583 There are two types of scrolling:

- 2584 1. The cursor moves with the data.
- 2585 2. The cursor remains stationary while the data moves.

2586 **3.336 Semaphore**

2587 A minimum synchronization primitive to serve as a basis for more complex synchronization
2588 mechanisms to be defined by the application program.

2589 **Note:** Semaphores are defined in detail in Section 4.15 (on page 104).

2590 **3.337 Session**

2591 A collection of process groups established for job control purposes. Each process group is a
2592 member of a session. A process is considered to be a member of the session of which its process
2593 group is a member. A newly created process joins the session of its creator. A process can alter
2594 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2595 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of
2596 IEEE Std 1003.1-2001.

2597 3.338 Session Leader

2598 A process that has created a session.

2599 **Note:** For further information, see the *setsid()* function defined in the System Interfaces volume of
2600 IEEE Std 1003.1-2001.

2601 3.339 Session Lifetime

2602 The period between when a session is created and the end of the lifetime of all the process
2603 groups that remain as members of the session.

2604 3.340 Shared Memory Object

2605 An object that represents memory that can be mapped concurrently into the address space of
2606 more than one process.

2607 3.341 Shell

2608 A program that interprets sequences of text input as commands. It may operate on an input
2609 stream or it may interactively prompt and read commands from a terminal.

2610 3.342 Shell, the

2611 The Shell Command Language Interpreter; a specific instance of a shell.

2612 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of
2613 IEEE Std 1003.1-2001.

2614 3.343 Shell Script

2615 A file containing shell commands. If the file is made executable, it can be executed by specifying
2616 its name as a simple command. Execution of a shell script causes a shell to execute the
2617 commands within the script. Alternatively, a shell can be requested to execute the commands in
2618 a shell script by specifying the name of the shell script as the operand to the *sh* utility.

2619 **Note:** Simple Commands are defined in detail in the Shell and Utilities volume of
2620 IEEE Std 1003.1-2001, Section 2.9.1, Simple Commands.

2621 The *sh* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

2622 3.344 Signal

2623 A mechanism by which a process or thread may be notified of, or affected by, an event occurring
2624 in the system. Examples of such events include hardware exceptions and specific actions by
2625 processes. The term signal is also used to refer to the event itself.

2626 3.345 Signal Stack

2627 Memory established for a thread, in which signal handlers catching signals sent to that thread
2628 are executed.

2629 3.346 Single-Quote

2630 The character ' ' , also known as apostrophe.

2631 3.347 Slash

2632 The character ' / ' , also known as solidus.

2633 3.348 Socket

2634 A file of a particular type that is used as a communications endpoint for process-to-process
2635 communication as described in the System Interfaces volume of IEEE Std 1003.1-2001.

2636 3.349 Socket Address

2637 An address associated with a socket or remote endpoint, including an address family identifier
2638 and addressing information specific to that address family. The address may include multiple
2639 parts, such as a network address associated with a host system and an identifier for a specific
2640 endpoint.

2641 3.350 Soft Limit

2642 A resource limitation established for each process that the process may set to any value less than
2643 or equal to the hard limit.

2644 3.351 Source Code

2645 When dealing with the Shell Command Language, input to the command language interpreter.
2646 The term “shell script” is synonymous with this meaning.

2647 When dealing with an ISO/IEC-conforming programming language, source code is input to a
2648 compiler conforming to that ISO/IEC standard.

2649 Source code also refers to the input statements prepared for the following standard utilities:
2650 *awk, bc, ed, lex, localedef, make, sed, and yacc.*

2651 Source code can also refer to a collection of sources meeting any or all of these meanings.

2652 **Note:** The *awk, bc, ed, lex, localedef, make, sed, and yacc* utilities are defined in detail in the Shell and
2653 Utilities volume of IEEE Std 1003.1-2001.

2654 **3.352 Space Character (<space>)**

2655 The character defined in the portable character set as <space>. The <space> is a member of the
 2656 **space** character class of the current locale, but represents the single character, and not all of the
 2657 possible members of the class; see also Section 3.431 (on page 95).

2658 **3.353 Spawn**

2659 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient
 2660 replacement for *fork()/exec*.

2661 **3.354 Special Built-In**

2662 See *Built-In Utility* in Section 3.83 (on page 46).

2663 **3.355 Special Parameter**

2664 In the shell command language, a parameter named by a single character from the following list:

2665 * @ # ? ! - \$ 0

2666 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
 2667 2.5.2, Special Parameters.

2668 **3.356 Spin Lock**

2669 A synchronization object used to allow multiple threads to serialize their access to shared data.

2670 **3.357 Sporadic Server**

2671 A scheduling policy for threads and processes that reserves a certain amount of execution
 2672 capacity for processing aperiodic events at a given priority level.

2673 **3.358 Standard Error**

2674 An output stream usually intended to be used for diagnostic messages.

2675 **3.359 Standard Input**

2676 An input stream usually intended to be used for primary data input.

2677 **3.360 Standard Output**

2678 An output stream usually intended to be used for primary data output.

2679 **3.361 Standard Utilities**

2680 The utilities described in the Shell and Utilities volume of IEEE Std 1003.1-2001.

2681 **3.362 Stream**

2682 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence
2683 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,
2684 *fopen()*, or *popen()* functions, and are associated with a file descriptor. A stream provides the
2685 additional services of user-selectable buffering and formatted input and output; see also Section
2686 3.363.

2687 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.5,
2688 Standard I/O Streams.

2689 The *fdopen()*, *fopen()*, or *popen()* functions are defined in detail in the System Interfaces volume
2690 of IEEE Std 1003.1-2001.

2691 **3.363 STREAM**

2692 Appearing in uppercase, STREAM refers to a full-duplex connection between a process and an
2693 open device or pseudo-device. It optionally includes one or more intermediate processing
2694 modules that are interposed between the process end of the STREAM and the device driver (or
2695 pseudo-device driver) end of the STREAM; see also Section 3.362.

2696 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.6,
2697 STREAMS.

2698 **3.364 STREAM End**

2699 The STREAM end is the driver end of the STREAM and is also known as the downstream end of
2700 the STREAM.

2701 **3.365 STREAM Head**

2702 The STREAM head is the beginning of the STREAM and is at the boundary between the system
2703 and the application process. This is also known as the upstream end of the STREAM.

2704 **3.366 STREAMS Multiplexor**

2705 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected above
2706 is referred to as N-to-1, or “upper multiplexing”. Multiplexing with STREAMS connected below
2707 is referred to as 1-to-N or “lower multiplexing”.

2708 **3.367 String**

2709 A contiguous sequence of bytes terminated by and including the first null byte.

2710 3.368 Subshell

2711 A shell execution environment, distinguished from the main or current shell execution
2712 environment.

2713 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.12,
2714 Shell Execution Environment.

2715 3.369 Successfully Transferred

2716 For a write operation to a regular file, when the system ensures that all data written is readable
2717 on any subsequent open of the file (even one that follows a system or power failure) in the
2718 absence of a failure of the physical storage medium.

2719 For a read operation, when an image of the data on the physical storage medium is available to
2720 the requesting process.

2721 3.370 Supplementary Group ID

2722 An attribute of a process used in determining file access permissions. A process has up to
2723 {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The
2724 supplementary group IDs of a process are set to the supplementary group IDs of the parent
2725 process when the process is created.

2726 3.371 Suspended Job

2727 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the
2728 process group to stop. A suspended job is a background job, but a background job is not
2729 necessarily a suspended job.

2730 3.372 Symbolic Link

2731 A type of file with the property that when the file is encountered during pathname resolution, a
2732 string stored by the file is used to modify the pathname resolution. The stored string has a length
2733 of {SYMLINK_MAX} bytes or fewer.

2734 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 102).

2735 3.373 Synchronized Input and Output

2736 A determinism and robustness improvement mechanism to enhance the data input and output
2737 mechanisms, so that an application can ensure that the data being manipulated is physically
2738 present on secondary mass storage devices.

2739 3.374 Synchronized I/O Completion

2740 The state of an I/O operation that has either been successfully transferred or diagnosed as
2741 unsuccessful.

2742 **3.375 Synchronized I/O Data Integrity Completion**

2743 For read, when the operation has been completed or diagnosed if unsuccessful. The read is
2744 complete only when an image of the data has been successfully transferred to the requesting
2745 process. If there were any pending write requests affecting the data to be read at the time that
2746 the synchronized read operation was requested, these write requests are successfully transferred
2747 prior to reading the data.

2748 For write, when the operation has been completed or diagnosed if unsuccessful. The write is
2749 complete only when the data specified in the write request is successfully transferred and all file
2750 system information required to retrieve the data is successfully transferred.

2751 File attributes that are not necessary for data retrieval (access time, modification time, status
2752 change time) need not be successfully transferred prior to returning to the calling process.

2753 **3.376 Synchronized I/O File Integrity Completion**

2754 Identical to a synchronized I/O data integrity completion with the addition that all file attributes
2755 relative to the I/O operation (including access time, modification time, status change time) are
2756 successfully transferred prior to returning to the calling process.

2757 **3.377 Synchronized I/O Operation**

2758 An I/O operation performed on a file that provides the application assurance of the integrity of
2759 its data and files.

2760 **3.378 Synchronous I/O Operation**

2761 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the
2762 processor until that I/O operation completes.

2763 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity completion or
2764 synchronized I/O file integrity completion.

2765 **3.379 Synchronously-Generated Signal**

2766 A signal that is attributable to a specific thread.

2767 For example, a thread executing an illegal instruction or touching invalid memory causes a
2768 synchronously-generated signal. Being synchronous is a property of how the signal was
2769 generated and not a property of the signal number.

2770 **3.380 System**

2771 An implementation of IEEE Std 1003.1-2001.

2772 3.381 System Crash

2773 An interval initiated by an unspecified circumstance that causes all processes (possibly other
2774 than special system processes) to be terminated in an undefined manner, after which any
2775 changes to the state and contents of files created or written to by an application prior to the
2776 interval are undefined, except as required elsewhere in IEEE Std 1003.1-2001.

2777 3.382 System Console

2778 An implementation-defined device that receives messages sent by the *syslog()* function, and the
2779 *fmtmsg()* function when the MM_CONSOLE flat is set.

2780 **Note:** The *syslog()* and *fmtmsg()* functions are defined in detail in the System Interfaces volume of
2781 IEEE Std 1003.1-2001.

2782 3.383 System Databases

2783 An implementation provides two system databases.

2784 The “group database” contains the following information for each group:

- 2785 1. Group name
- 2786 2. Numerical group ID
- 2787 3. List of all users allowed in the group

2788 The “user database” contains the following information for each user:

- 2789 1. User name
- 2790 2. Numerical user ID
- 2791 3. Numerical group ID
- 2792 4. Initial working directory
- 2793 5. Initial user program

2794 If the initial user program field is null, the system default is used. If the initial working directory
2795 field is null, the interpretation of that field is implementation-defined. These databases may
2796 contain other fields that are unspecified by IEEE Std 1003.1-2001.

2797 3.384 System Documentation

2798 All documentation provided with an implementation except for the conformance document.
2799 Electronically distributed documents for an implementation are considered part of the system
2800 documentation.

2801 3.385 System Process

2802 An implementation-defined object, other than a process executing an application, that has a
2803 process ID.

2804 3.386 System Reboot

2805 An implementation-defined sequence of events that may result in the loss of transitory data; that
2806 is, data that is not saved in permanent storage. For example, message queues, shared memory,
2807 semaphores, and processes.

2808 3.387 System Trace Event

2809 A trace event that is generated by the implementation, in response either to a system-initiated
2810 action or to an application-requested action, except for a call to *posix_trace_event()*. When
2811 supported by the implementation, a system-initiated action generates a process-independent
2812 system trace event and an application-requested action generates a process-dependent system
2813 trace event. For a system trace event not defined by IEEE Std 1003.1-2001, the associated trace
2814 event type identifier is derived from the implementation-defined name for this trace event, and
2815 the associated data is of implementation-defined content and length.

2816 3.388 System-Wide

2817 Pertaining to events occurring in all processes existing in an implementation at a given point in
2818 time.

2819 3.389 Tab Character (<tab>)

2820 A character that in the output stream indicates that printing or displaying should start at the
2821 next horizontal tabulation position on the current line. It is the character designated by '*\t*' in
2822 the C language. If the current position is at or past the last defined horizontal tabulation
2823 position, the behavior is unspecified. It is unspecified whether this character is the exact
2824 sequence transmitted to an output device by the system to accomplish the tabulation.

2825 3.390 Terminal (or Terminal Device)

2826 A character special file that obeys the specifications of the general terminal interface.

2827 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 187).

2828 3.391 Text Column

2829 A roughly rectangular block of characters capable of being laid out side-by-side next to other
2830 text columns on an output page or terminal screen. The widths of text columns are measured in
2831 column positions.

2832 3.392 Text File

2833 A file that contains characters organized into one or more lines. The lines do not contain NUL
 2834 characters and none can exceed {LINE_MAX} bytes in length, including the <newline>.
 2835 Although IEEE Std 1003.1-2001 does not distinguish between text files and binary files (see the
 2836 ISO C standard), many utilities only produce predictable or meaningful output when operating
 2837 on text files. The standard utilities that have such restrictions always specify “text files” in their
 2838 STDIN or INPUT FILES sections.

2839 3.393 Thread

2840 A single flow of control within a process. Each thread has its own thread ID, scheduling priority
 2841 and policy, *errno* value, thread-specific key/value bindings, and the required system resources to
 2842 support a flow of control. Anything whose address may be determined by a thread, including
 2843 but not limited to static variables, storage obtained via *malloc()*, directly addressable storage
 2844 obtained through implementation-defined functions, and automatic variables, are accessible to
 2845 all threads in the same process.

2846 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of
 2847 IEEE Std 1003.1-2001.

2848 3.394 Thread ID

2849 Each thread in a process is uniquely identified during its lifetime by a value of type **pthread_t**
 2850 called a thread ID.

2851 3.395 Thread List

2852 An ordered set of runnable threads that all have the same ordinal value for their priority.

2853 The ordering of threads on the list is determined by a scheduling policy or policies. The set of
 2854 thread lists includes all runnable threads in the system.

2855 3.396 Thread-Safe

2856 A function that may be safely invoked concurrently by multiple threads. Each function defined
 2857 in the System Interfaces volume of IEEE Std 1003.1-2001 is thread-safe unless explicitly stated
 2858 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is
 2859 accessing static storage, or objects shared among threads.

2860 3.397 Thread-Specific Data Key

2861 A process global handle of type **pthread_key_t** which is used for naming thread-specific data.

2862 Although the same key value may be used by different threads, the values bound to the key by
 2863 *pthread_setspecific()* and accessed by *pthread_getspecific()* are maintained on a per-thread basis
 2864 and persist for the life of the calling thread.

2865 **Note:** The *pthread_getspecific()* and *pthread_setspecific()* functions are defined in detail in the System
 2866 Interfaces volume of IEEE Std 1003.1-2001.

2867 **3.398 Tilde**

2868 The character '~'.

2869 **3.399 Timeouts**

2870 A method of limiting the length of time an interface will block; see also Section 3.76 (on page 45).

2871 **3.400 Timer**2872 A mechanism that can notify a thread when the time as measured by a particular clock has
2873 reached or passed a specified value, or when a specified amount of time has passed.2874 **3.401 Timer Overrun**2875 A condition that occurs each time a timer, for which there is already an expiration signal queued
2876 to the process, expires.2877 **3.402 Token**2878 In the shell command language, a sequence of characters that the shell considers as a single unit
2879 when reading input. A token is either an operator or a word.2880 **Note:** The rules for reading input are defined in detail in the Shell and Utilities volume of
2881 IEEE Std 1003.1-2001, Section 2.3, Token Recognition.2882 **3.403 Trace Analyzer Process**2883 A process that extracts trace events from a trace stream to retrieve information about the
2884 behavior of an application.2885 **3.404 Trace Controller Process**

2886 A process that creates a trace stream for tracing a process.

2887 **3.405 Trace Event**2888 A data object that represents an action executed by the system, and that is recorded in a trace
2889 stream.2890 **3.406 Trace Event Type**

2891 A data object type that defines a class of trace event.

2892 3.407 Trace Event Type Mapping

2893 A one-to-one mapping between trace event types and trace event names.

2894 3.408 Trace Filter

2895 A filter that allows the trace controller process to specify those trace event types that are to be
2896 ignored; that is, not generated.

2897 3.409 Trace Generation Version

2898 A data object that is an implementation-defined character string, generated by the trace system
2899 and describing the origin and version of the trace system.

2900 3.410 Trace Log

2901 The flushed image of a trace stream, if the trace stream is created with a trace log.

2902 3.411 Trace Point

2903 An action that may cause a trace event to be generated.

2904 3.412 Trace Stream

2905 An opaque object that contains trace events plus internal data needed to interpret those trace
2906 events.

2907 3.413 Trace Stream Identifier

2908 A handle to manage tracing operations in a trace stream.

2909 3.414 Trace System

2910 A system that allows both system and user trace events to be generated into a trace stream.
2911 These trace events can be retrieved later.

2912 3.415 Traced Process

2913 A process for which at least one trace stream has been created. A traced process is also called a
2914 target process.

2915 3.416 Tracing Status of a Trace Stream

2916 A status that describes the state of an active trace stream. The tracing status of a trace stream can
2917 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:
2918 running or suspended.

2919 3.417 Typed Memory Name Space

2920 A system-wide name space that contains the names of the typed memory objects present in the
2921 system. It is configurable for a given implementation.

2922 3.418 Typed Memory Object

2923 A combination of a typed memory pool and a typed memory port. The entire contents of the
2924 pool are accessible from the port. The typed memory object is identified through a name that
2925 belongs to the typed memory name space.

2926 3.419 Typed Memory Pool

2927 An extent of memory with the same operational characteristics. Typed memory pools may be
2928 contained within each other.

2929 3.420 Typed Memory Port

2930 A hardware access path to one or more typed memory pools.

2931 3.421 Unbind

2932 Remove the association between a network address and an endpoint.

2933 3.422 Unit Data

2934 See *Datagram* in Section 3.123 (on page 52).

2935 3.423 Upshifting

2936 The conversion of a lowercase character that has a single-character uppercase representation
2937 into this uppercase representation.

2938 **3.424 User Database**

2939 A system database of implementation-defined format that contains at least the following
2940 information for each user ID:

- 2941 • User name
- 2942 • Numerical user ID
- 2943 • Initial numerical group ID
- 2944 • Initial working directory
- 2945 • Initial user program

2946 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under
2947 which the initial values are operative are implementation-defined.

2948 If the initial user program field is null, an implementation-defined program is used.

2949 If the initial working directory field is null, the interpretation of that field is implementation-
2950 defined.

2951 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-2001.

2952 **3.425 User ID**

2953 A non-negative integer that is used to identify a system user. When the identity of a user is
2954 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a
2955 saved set-user-ID.

2956 **3.426 User Name**

2957 A string that is used to identify a user; see also Section 3.424. To be portable across systems
2958 conforming to IEEE Std 1003.1-2001, the value is composed of characters from the portable
2959 filename character set. The hyphen should not be used as the first character of a portable user
2960 name.

2961 **3.427 User Trace Event**

2962 A trace event that is generated explicitly by the application as a result of a call to
2963 *posix_trace_event()*.

2964 **3.428 Utility**

2965 A program, excluding special built-in utilities provided as part of the Shell Command Language,
2966 that can be called by name from a shell to perform a specific task, or related set of tasks.

2967 **Note:** For further information on special built-in utilities, see the Shell and Utilities volume of
2968 IEEE Std 1003.1-2001, Section 2.14, Special Built-In Utilities.

2969 **3.429 Variable**

2970 In the shell command language, a named parameter.

2971 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.5,
2972 Parameters and Variables.

2973 **3.430 Vertical-Tab Character (<vertical-tab>)**

2974 A character that in the output stream indicates that printing should start at the next vertical
2975 tabulation position. It is the character designated by '`\v`' in the C language. If the current
2976 position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is
2977 unspecified whether this character is the exact sequence transmitted to an output device by the
2978 system to accomplish the tabulation.

2979 **3.431 White Space**

2980 A sequence of one or more characters that belong to the **space** character class as defined via the
2981 `LC_CTYPE` category in the current locale.

2982 In the POSIX locale, white space consists of one or more `<blank>s` (`<space>s` and `<tab>s`),
2983 `<newline>s`, `<carriage-return>s`, `<form-feed>s`, and `<vertical-tab>s`.

2984 **3.432 Wide-Character Code (C Language)**

2985 An integer value corresponding to a single graphic symbol or control code.

2986 **Note:** C Language Wide-Character Codes are defined in detail in Section 6.3 (on page 119).

2987 **3.433 Wide-Character Input/Output Functions**

2988 The functions that perform wide-oriented input from streams or wide-oriented output to
2989 streams: `fgetwc()`, `fgetws()`, `fputwc()`, `fputws()`, `fwprintf()`, `fwscanf()`, `getwc()`, `getwchar()`, `putwc()`,
2990 `putwchar()`, `ungetwc()`, `vfwprintf()`, `vfwscanf()`, `vwprintf()`, `vwscanf()`, `wprintf()`, and `wscanf()`.

2991 **Note:** These functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-2001.

2992 **3.434 Wide-Character String**

2993 A contiguous sequence of wide-character codes terminated by and including the first null wide-
2994 character code.

2995 3.435 Word

2996 In the shell command language, a token other than an operator. In some cases a word is also a
2997 portion of a word token: in the various forms of parameter expansion, such as `${name-word}`, and
2998 variable assignment, such as `name=word`, the word is the portion of the token depicted by *word*.
2999 The concept of a word is no longer applicable following word expansions—only fields remain.

3000 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
3001 2.6.2, Parameter Expansion and the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
3002 2.6, Word Expansions.

3003 3.436 Working Directory (or Current Working Directory)

3004 A directory, associated with a process, that is used in pathname resolution for pathnames that
3005 do not begin with a slash.

3006 3.437 Worldwide Portability Interface

3007 Functions for handling characters in a codeset-independent manner.

3008 3.438 Write

3009 To output characters to a file, such as standard output or standard error. Unless otherwise
3010 stated, standard output is the default output destination for all uses of the term “write”; see the
3011 distinction between display and write in Section 3.132 (on page 53).

3012 3.439 XSI

3013 The X/Open System Interface is the core application programming interface for C and *sh*
3014 programming for systems conforming to the Single UNIX Specification. This is a superset of the
3015 mandatory requirements for conformance to IEEE Std 1003.1-2001.

3016 3.440 XSI-Conformant

3017 A system which allows an application to be built using a set of services that are consistent across
3018 all systems that conform to IEEE Std 1003.1-2001 and that support the XSI extension.

3019 **Note:** See also Chapter 2 (on page 17).

3020 3.441 Zombie Process

3021 A process that has terminated and that is deleted when its exit status has been reported to
3022 another process which is waiting for that process to terminate.

3023 **3.442** ± 0

3024 The algebraic sign provides additional information about any variable that has the value zero
3025 when the representation allows the sign to be determined.

General Concepts

3027

3028 For the purposes of IEEE Std 1003.1-2001, the general concepts given in Chapter 4 apply.

3029 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and
 3030 definitions given in this chapter are used elsewhere in text related to extensions and options,
 3031 they are shaded as appropriate.

3032 4.1 Concurrent Execution

3033 Functions that suspend the execution of the calling thread shall not cause the execution of other
 3034 threads to be indefinitely suspended.

3035 4.2 Directory Protection

3036 If a directory is writable and the mode bit `S_ISVTX` is set on the directory, a process may remove
 3037 or rename files within that directory only if one or more of the following is true:

- 3038 • The effective user ID of the process is the same as that of the owner ID of the file.
- 3039 • The effective user ID of the process is the same as that of the owner ID of the directory.
- 3040 • The process has appropriate privileges.

3041 If the `S_ISVTX` bit is set on a non-directory file, the behavior is unspecified.

3042 4.3 Extended Security Controls

3043 An implementation may provide implementation-defined extended security controls (see
 3044 Section 3.159 (on page 57)). These permit an implementation to provide security mechanisms to
 3045 implement different security policies than those described in IEEE Std 1003.1-2001. These
 3046 mechanisms shall not alter or override the defined semantics of any of the interfaces in
 3047 IEEE Std 1003.1-2001.

3048 4.4 File Access Permissions

3049 The standard file access control mechanism uses the file permission bits, as described below.

3050 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An
 3051 additional access control mechanism shall only further restrict the access permissions defined by
 3052 the file permission bits. An alternate file access control mechanism shall:

- 3053 • Specify file permission bits for the file owner class, file group class, and file other class of that
 3054 file, corresponding to the access permissions.
- 3055 • Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the
 3056 appropriate privilege.
- 3057 • Be disabled for a file after the file permission bits are changed for that file with `chmod()`. The
 3058 disabling of the alternate mechanism need not disable any additional mechanisms supported

3059 by an implementation.

3060 Whenever a process requests file access permission for read, write, or execute/search, if no
3061 additional mechanism denies access, access shall be determined as follows:

- 3062 • If a process has the appropriate privilege:
 - 3063 — If read, write, or directory search permission is requested, access shall be granted.
 - 3064 — If execute permission is requested, access shall be granted if execute permission is
3065 granted to at least one user by the file permission bits or by an alternate access control
3066 mechanism; otherwise, access shall be denied.
- 3067 • Otherwise:
 - 3068 — The file permission bits of a file contain read, write, and execute/search permissions for
3069 the file owner class, file group class, and file other class.
 - 3070 — Access shall be granted if an alternate access control mechanism is not enabled and the
3071 requested access permission bit is set for the class (file owner class, file group class, or file
3072 other class) to which the process belongs, or if an alternate access control mechanism is
3073 enabled and it allows the requested access; otherwise, access shall be denied.

3074 4.5 File Hierarchy

3075 Files in the system are organized in a hierarchical structure in which all of the non-terminal
3076 nodes are directories and all of the terminal nodes are any other type of file. Since multiple
3077 directory entries may refer to the same file, the hierarchy is properly described as a “directed
3078 graph”.

3079 4.6 Filenames

3080 For a filename to be portable across implementations conforming to IEEE Std 1003.1-2001, it
3081 shall consist only of the portable filename character set as defined in Section 3.276 (on page 73).

3082 The hyphen character shall not be used as the first character of a portable filename. Uppercase
3083 and lowercase letters shall retain their unique identities between conforming implementations.
3084 In the case of a portable pathname, the slash character may also be used.

3085 4.7 File Times Update

3086 Each file has three distinct associated time values: *st_atime*, *st_mtime*, and *st_ctime*. The *st_atime*
3087 field is associated with the times that the file data is accessed; *st_mtime* is associated with the
3088 times that the file data is modified; and *st_ctime* is associated with the times that the file status is
3089 changed. These values are returned in the file characteristics structure, as described in
3090 **<sys/stat.h>**.

3091 Each function or utility in IEEE Std 1003.1-2001 that reads or writes data or changes file status
3092 indicates which of the appropriate time-related fields shall be “marked for update”. If an
3093 implementation of such a function or utility marks for update a time-related field not specified
3094 by IEEE Std 1003.1-2001, this shall be documented, except that any changes caused by pathname
3095 resolution need not be documented. For the other functions or utilities in IEEE Std 1003.1-2001
3096 (those that are not explicitly required to read or write file data or change file status, but that in
3097 some implementations happen to do so), the effect is unspecified.

3098 An implementation may update fields that are marked for update immediately, or it may update
3099 such fields periodically. At an update point in time, any marked fields shall be set to the current
3100 time and the update marks shall be cleared. All fields that are marked for update shall be
3101 updated when the file ceases to be open by any process, or when a *stat()*, *fstat()*, or *lstat()* is
3102 performed on the file. Other times at which updates are done are unspecified. Marks for update,
3103 and updates themselves, are not done for files on read-only file systems; see Section 3.304 (on
3104 page 77).

3105 4.8 Host and Network Byte Orders

3106 When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned
3107 values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be
3108 stored in several octets. The convention is that all such values are stored with 8 bits in each octet,
3109 and with the first (lowest-addressed) octet holding the most-significant bits. This is called
3110 “network byte order”.

3111 Network byte order may not be convenient for processing actual values. For this, it is more
3112 sensible for values to be stored as ordinary integers. This is known as “host byte order”. In host
3113 byte order:

- 3114 • The most significant bit might not be stored in the first byte in address order.
- 3115 • Bits might not be allocated to bytes in any obvious order at all.

3116 8-bit values stored in **uint8_t** objects do not require conversion to or from host byte order, as
3117 they have the same representation. 16 and 32-bit values can be converted using the *htonl()*,
3118 *htons()*, *ntohl()*, and *ntohs()* functions. When reading data that is to be converted to host byte
3119 order, it should either be received directly into a **uint16_t** or **uint32_t** object or should be copied
3120 from an array of bytes using *memcpy()* or similar. Passing the data through other types could
3121 cause the byte order to be changed. Similar considerations apply when sending data.

3122 4.9 Measurement of Execution Time

3123 The mechanism used to measure execution time shall be implementation-defined. The
3124 implementation shall also define to whom the CPU time that is consumed by interrupt handlers
3125 and system services on behalf of the operating system will be charged. See Section 3.117 (on
3126 page 51).

4.10 Memory Synchronization

Applications shall ensure that access to any memory location by more than one thread of control (threads or processes) is restricted such that no thread of control can read or modify a memory location while another thread of control may be modifying it. Such access is restricted using functions that synchronize thread execution and also synchronize memory with respect to other threads. The following functions synchronize memory with respect to other threads:

<code>fork()</code>	<code>pthread_mutex_timedlock()</code>	<code>pthread_rwlock_tryrdlock()</code>
<code>pthread_barrier_wait()</code>	<code>pthread_mutex_trylock()</code>	<code>pthread_rwlock_trywrlock()</code>
<code>pthread_cond_broadcast()</code>	<code>pthread_mutex_unlock()</code>	<code>pthread_rwlock_unlock()</code>
<code>pthread_cond_signal()</code>	<code>pthread_spin_lock()</code>	<code>pthread_rwlock_wrlock()</code>
<code>pthread_cond_timedwait()</code>	<code>pthread_spin_trylock()</code>	<code>sem_post()</code>
<code>pthread_cond_wait()</code>	<code>pthread_spin_unlock()</code>	<code>sem_trywait()</code>
<code>pthread_create()</code>	<code>pthread_rwlock_rdlock()</code>	<code>sem_wait()</code>
<code>pthread_join()</code>	<code>pthread_rwlock_timedrdlock()</code>	<code>wait()</code>
<code>pthread_mutex_lock()</code>	<code>pthread_rwlock_timedwrlock()</code>	<code>waitpid()</code>

The `pthread_once()` function shall synchronize memory for the first call in each thread for a given **pthread_once_t** object.

Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified whether the invocation causes memory to be synchronized.

Applications may allow more than one thread of control to read a memory location simultaneously.

4.11 Pathname Resolution

Pathname resolution is performed for a process to resolve a pathname to a particular file in a file hierarchy. There may be multiple pathnames that resolve to the same file.

Each filename in the pathname is located in the directory specified by its predecessor (for example, in the pathname fragment **a/b**, file **b** is located in directory **a**). Pathname resolution shall fail if this cannot be accomplished. If the pathname begins with a slash, the predecessor of the first filename in the pathname shall be taken to be the root directory of the process (such pathnames are referred to as “absolute pathnames”). If the pathname does not begin with a slash, the predecessor of the first filename of the pathname shall be taken to be the current working directory of the process (such pathnames are referred to as “relative pathnames”).

The interpretation of a pathname component is dependent on the value of `{NAME_MAX}` and `_POSIX_NO_TRUNC` associated with the path prefix of that component. If any pathname component is longer than `{NAME_MAX}`, the implementation shall consider this an error.

A pathname that contains at least one non-slash character and that ends with one or more trailing slashes shall be resolved as if a single dot character (`'.'`) were appended to the pathname.

If a symbolic link is encountered during pathname resolution, the behavior shall depend on whether the pathname component is at the end of the pathname and on the function being performed. If all of the following are true, then pathname resolution is complete:

1. This is the last pathname component of the pathname.
2. The pathname has no trailing slash.

3169 3. The function is required to act on the symbolic link itself, or certain arguments direct that
3170 the function act on the symbolic link itself.

3171 In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the
3172 symbolic link. If the combined length exceeds {PATH_MAX}, and the implementation considers
3173 this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication shall be
3174 returned. Otherwise, the resolved pathname shall be the resolution of the pathname just created.
3175 If the resulting pathname does not begin with a slash, the predecessor of the first filename of the
3176 pathname is taken to be the directory containing the symbolic link.

3177 If the system detects a loop in the pathname resolution process, it shall set *errno* to [ELOOP] and
3178 return an error indication. The same may happen if during the resolution process more symbolic
3179 links were followed than the implementation allows. This implementation-defined limit shall
3180 not be smaller than {SYMLOOP_MAX}.

3181 The special filename dot shall refer to the directory specified by its predecessor. The special
3182 filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case,
3183 in the root directory, dot-dot may refer to the root directory itself.

3184 A pathname consisting of a single slash shall resolve to the root directory of the process. A null
3185 pathname shall not be successfully resolved. A pathname that begins with two successive
3186 slashes may be interpreted in an implementation-defined manner, although more than two
3187 leading slashes shall be treated as a single slash.

3188 **4.12 Process ID Reuse**

3189 A process group ID shall not be reused by the system until the process group lifetime ends.

3190 A process ID shall not be reused by the system until the process lifetime ends. In addition, if
3191 there exists a process group whose process group ID is equal to that process ID, the process ID
3192 shall not be reused by the system until the process group lifetime ends. A process that is not a
3193 system process shall not have a process ID of 1.

3194 **4.13 Scheduling Policy**

3195 A scheduling policy affects process or thread ordering:

- 3196 • When a process or thread is a running thread and it becomes a blocked thread
- 3197 • When a process or thread is a running thread and it becomes a preempted thread
- 3198 • When a process or thread is a blocked thread and it becomes a runnable thread
- 3199 • When a running thread calls a function that can change the priority or scheduling policy of a
3200 process or thread
- 3201 • In other scheduling policy-defined circumstances

3202 Conforming implementations shall define the manner in which each of the scheduling policies
3203 may modify the priorities or otherwise affect the ordering of processes or threads at each of the
3204 occurrences listed above. Additionally, conforming implementations shall define in what other
3205 circumstances and in what manner each scheduling policy may modify the priorities or affect
3206 the ordering of processes or threads.

4.14 Seconds Since the Epoch

A value that approximates the number of seconds that have elapsed since the Epoch. A Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*), hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the expression below.

If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and the value is non-negative, the value is related to a Coordinated Universal Time name according to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all integer types:

$$tm\_sec + tm\_min*60 + tm\_hour*3600 + tm\_yday*86400 + \\ (tm\_year-70)*31536000 + ((tm\_year-69)/4)*86400 - \\ ((tm\_year-1)/100)*86400 + ((tm\_year+299)/400)*86400$$

The relationship between the actual time of day and the current value for seconds since the Epoch is unspecified.

How any changes to the value of seconds since the Epoch are made to align to a desired relationship with the current actual time are made is implementation-defined. As represented in seconds since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

Note: The last three terms of the expression add in a day for each year that follows a leap year starting with the first leap year since the Epoch. The first term adds a day every 4 years starting in 1973, the second subtracts a day back out every 100 years starting in 2001, and the third adds a day back in every 400 years starting in 2001. The divisions in the formula are integer divisions; that is, the remainder is discarded leaving only the integer quotient.

4.15 Semaphore

A minimum synchronization primitive to serve as a basis for more complex synchronization mechanisms to be defined by the application program.

For the semaphores associated with the Semaphores option, a semaphore is represented as a shareable resource that has a non-negative integer value. When the value is zero, there is a (possibly empty) set of threads awaiting the availability of the semaphore.

For the semaphores associated with the X/Open System Interface Extension (XSI), a semaphore is a positive integer (0 through 32767). The *semget()* function can be called to create a set or array of semaphores. A semaphore set can contain one or more semaphores up to an implementation-defined value.

Semaphore Lock Operation

An operation that is applied to a semaphore. If, prior to the operation, the value of the semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented.

3244 Semaphore Unlock Operation

3245 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in
 3246 the set of threads awaiting the semaphore, then some thread from that set shall be removed from
 3247 the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

3248 4.16 Thread-Safety

3249 Refer to the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.9, Threads.

3250 4.17 Tracing

3251 The trace system allows a traced process to have a selection of events created for it. Traces
 3252 consist of streams of trace event types.

3253 A trace event type is identified on the one hand by a trace event type name, also referenced as a
 3254 trace event name, and on the other hand by a trace event type identifier. A trace event name is a
 3255 human-readable string. A trace event type identifier is an opaque identifier used by the trace
 3256 system. There shall be a one-to-one relationship between trace event type identifiers and trace
 3257 event names for a given trace stream and also for a given traced process. The trace event type
 3258 identifier shall be generated automatically from a trace event name by the trace system either
 3259 when a trace controller process invokes *posix_trace_trid_eventid_open()* or when an instrumented
 3260 application process invokes *posix_trace_eventid_open()*. Trace event type identifiers are used to
 3261 filter trace event types, to allow interpretation of user data, and to identify the kind of trace point
 3262 that generated a trace event.

3263 Each trace event shall be of a particular trace event type, and associated with a trace event type
 3264 identifier. The execution of a trace point shall generate a trace event if a trace stream has been
 3265 created and started for the process that executed the trace point and if the corresponding trace
 3266 event type identifier is not ignored by filtering.

3267 A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a
 3268 trace log is associated with the trace stream, except that:

- 3269 • For a trace stream, if no resources are available for the event, the event is lost.
- 3270 • For a trace log, if no resources are available for the event, or a flush operation does not
 3271 succeed, the event is lost.

3272 A trace event recorded in an active trace stream may be retrieved by an application having the
 3273 appropriate privileges.

3274 A trace event recorded in a trace log may be retrieved by an application having the appropriate
 3275 privileges after opening the trace log as a pre-recorded trace stream, with the function
 3276 *posix_trace_open()*.

3277 When a trace event is reported it is possible to retrieve the following:

- 3278 • A trace event type identifier
- 3279 • A timestamp
- 3280 • The process ID of the traced process, if the trace event is process-dependent
- 3281 • Any optional trace event data including its length

- 3282 • If the Threads option is supported, the thread ID, if the trace event is process-dependent
- 3283 • The program address at which the trace point was invoked

3284 Trace events may be mapped from trace event types to trace event names. One such mapping
 3285 shall be associated with each trace stream. An active trace stream is associated with a traced
 3286 process, and also with its children if the Trace Inherit option is supported and also the
 3287 inheritance policy is set to `_POSIX_TRACE_INHERIT`. Therefore each traced process has a
 3288 mapping of the trace event names to trace event type identifiers that have been defined for that
 3289 process.

3290 Traces can be recorded into either trace streams or trace logs.

3291 The implementation and format of a trace stream are unspecified. A trace stream need not be
 3292 and generally is not persistent. A trace stream may be either active or pre-recorded:

- 3293 • An active trace stream is a trace stream that has been created and has not yet been shut
 3294 down. It can be of one of the two following classes:
 - 3295 1. An active trace stream without a trace log that was created with the *posix_trace_create()*
 3296 function
 - 3297 2. If the Trace Log option is supported, an active trace stream with a trace log that was
 3298 created with the *posix_trace_create_withlog()* function
- 3299 • A pre-recorded trace stream is a trace stream that was opened from a trace log object using
 3300 the *posix_trace_open()* function.

3301 An active trace stream can loop. This behavior means that when the resources allocated by the
 3302 trace system for the trace stream are exhausted, the trace system reuses the resources associated
 3303 with the oldest recorded trace events to record new trace events.

3304 If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This
 3305 operation causes the trace system to write trace events from the trace stream to the associated
 3306 trace log, following the defined policies or using an explicit function call. After this operation,
 3307 the trace system may reuse the resources associated with the flushed trace events.

3308 An active trace stream with or without a trace log can be cleared. This operation shall cause all
 3309 the resources associated with this trace stream to be reinitialized. The trace stream shall behave
 3310 as if it was returning from its creation, except that the mapping of trace event type identifiers to
 3311 trace event names shall not be cleared. If a trace log was associated with this trace stream, the
 3312 trace log shall also be reinitialized.

3313 A trace log shall be recorded when the *posix_trace_shutdown()* operation is invoked or during
 3314 tracing, depending on the tracing strategy which is defined by a log policy. After the trace
 3315 stream has been shut down, the trace information can be retrieved from the associated trace log
 3316 using the same interface used to retrieve information from an active trace stream.

3317 For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance
 3318 attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process shall be traced together
 3319 with all of its future children. The *posix_pid* member of each trace event in a trace stream shall be
 3320 the process ID of the traced process.

3321 Each trace point may be an implementation-defined action such as a context switch, or an
 3322 application-programmed action such as a call to a specific operating system service (for
 3323 example, *fork()*) or a call to *posix_trace_event()*.

3324 Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace
 3325 events. By default, no trace events are filtered.

3326 The results of the tracing operations can be analyzed and monitored by a trace controller process
3327 or a trace analyzer process.

3328 Only the trace controller process has control of the trace stream it has created. The control of the
3329 operation of a trace stream is done using its corresponding trace stream identifier. The trace
3330 controller process is able to:

- 3331 • Initialize the attributes of a trace stream
- 3332 • Create the trace stream
- 3333 • Start and stop tracing
- 3334 • Know the mapping of the traced process
- 3335 • If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- 3336 • Shut the trace stream down

3337 A traced process may also be a trace controller process. Only the trace controller process can
3338 control its trace stream(s). A trace stream created by a trace controller process shall be shut
3339 down if its controller process terminates or executes another file.

3340 A trace controller process may also be a trace analyzer process. Trace analysis can be done
3341 concurrently with the traced process or can be done off-line, in the same or in a different
3342 platform.

3343 **4.18 Treatment of Error Conditions for Mathematical Functions**

3344 For all the functions in the `<math.h>` header, an application wishing to check for error situations
3345 should set `errno` to 0 and call `feclearexcept(FE_ALL_EXCEPT)` before calling the function. On
3346 return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW |`
3347 `FE_UNDERFLOW)` is non-zero, an error has occurred.

3348 The following error conditions are defined for all functions in the `<math.h>` header.

3349 **4.18.1 Domain Error**

3350 A “domain error” shall occur if an input argument is outside the domain over which the
3351 mathematical function is defined. The description of each function lists any required domain
3352 errors; an implementation may define additional domain errors, provided that such errors are
3353 consistent with the mathematical definition of the function.

3354 On a domain error, the function shall return an implementation-defined value; if the integer
3355 expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to [EDOM]; if
3356 the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “invalid”
3357 floating-point exception shall be raised.

3358 **4.18.2 Pole Error**

3359 A “pole error” occurs if the mathematical result of the function is an exact infinity (for example,
3360 $\log(0.0)$).

3361 On a pole error, the function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or
3362 `HUGE_VALL` according to the return type, with the same sign as the correct value of the
3363 function; if the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero, *errno* shall
3364 be set to `[ERANGE]`; if the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is
3365 non-zero, the “divide-by-zero” floating-point exception shall be raised.

3366 **4.18.3 Range Error**

3367 A “range error” shall occur if the finite mathematical result of the function cannot be
3368 represented in an object of the specified type, due to extreme magnitude.

3369 **4.18.3.1 Result Overflows**

3370 A floating result overflows if the magnitude of the mathematical result is finite but so large that
3371 the mathematical result cannot be represented without extraordinary roundoff error in an object
3372 of the specified type. If a floating result overflows and default rounding is in effect, then the
3373 function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or `HUGE_VALL`
3374 according to the return type, with the same sign as the correct value of the function; if the integer
3375 expression (`math_errhandling` & `MATH_ERRNO`) is non-zero, *errno* shall be set to `[ERANGE]`; if
3376 the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the “overflow”
3377 floating-point exception shall be raised.

3378 **4.18.3.2 Result Underflows**

3379 The result underflows if the magnitude of the mathematical result is so small that the
3380 mathematical result cannot be represented, without extraordinary roundoff error, in an object of
3381 the specified type. If the result underflows, the function shall return an implementation-defined
3382 value whose magnitude is no greater than the smallest normalized positive number in the
3383 specified type; if the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,
3384 whether *errno* is set to `[ERANGE]` is implementation-defined; if the integer expression
3385 (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, whether the “underflow” floating-point
3386 exception is raised is implementation-defined.

3387 **4.19 Treatment of NaN Arguments for the Mathematical Functions**

3388 For functions called with a NaN argument, no errors shall occur and a NaN shall be returned,
3389 except where stated otherwise.

3390 If a function with one or more NaN arguments returns a NaN result, the result should be the
3391 same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

3392 On implementations that support the IEC 60559:1989 standard floating point, functions with
3393 signaling NaN argument(s) shall be treated as if the function were called with an argument that
3394 is a required domain error and shall return a quiet NaN result, except where stated otherwise.

3395 **Note:** The function might never see the signaling NaN, since it might trigger when the arguments are
3396 evaluated during the function call.

3397 On implementations that support the IEC 60559:1989 standard floating point, for those
3398 functions that do not have a documented domain error, the following shall apply:

3399 These functions shall fail if:

3400 Domain Error Any argument is a signaling NaN.

3401 Either, the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero and *errno*

3402 shall be set to [EDOM], or the integer expression (`math_errhandling & MATH_ERREXCEPT`)

3403 is non-zero and the invalid floating-point exception shall be raised.

3404 4.20 Utility

3405 A utility program shall be either an executable file, such as might be produced by a compiler or

3406 linker system from computer source code, or a file of shell source code, directly interpreted by

3407 the shell. The program may have been produced by the user, provided by the system

3408 implementor, or acquired from an independent distributor.

3409 The system may implement certain utilities as shell functions (see the Shell and Utilities volume

3410 of IEEE Std 1003.1-2001, Section 2.9.5, Function Definition Command) or built-in utilities, but

3411 only an application that is aware of the command search order described in the Shell and

3412 Utilities volume of IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution or of

3413 performance characteristics can discern differences between the behavior of such a function or

3414 built-in utility and that of an executable file.

3415 4.21 Variable Assignment

3416 In the shell command language, a word consisting of the following parts:

3417 `varname=value`

3418 When used in a context where assignment is defined to occur and at no other time, the *value*

3419 (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

3420 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section

3421 2.9.1, Simple Commands.

3422 The *varname* and *value* parts shall meet the requirements for a name and a word, respectively,

3423 except that they are delimited by the embedded unquoted equals-sign, in addition to other

3424 delimiters.

3425 **Note:** Additional delimiters are described in the Shell and Utilities volume of IEEE Std 1003.1-2001,

3426 Section 2.3, Token Recognition.

3427 When a variable assignment is done, the variable shall be created if it did not already exist. If

3428 *value* is not specified, the variable shall be given a null value.

3429 **Note:** An alternative form of variable assignment:

3430 `symbol=value`

3431 (where *symbol* is a valid word delimited by an equals-sign, but not a valid name) produces

3432 unspecified results. The form `symbol=value` is used by the KornShell `name[expression]=value`

3433 syntax.

File Format Notation

3435

3436 The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility
 3437 descriptions use a syntax to describe the data organization within the files, when that
 3438 organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces
 3439 volume of IEEE Std 1003.1-2001 *printf()* function, as described in this chapter. When used in
 3440 STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that
 3441 could have been used to write the text to be read, not a format that could be used by the System
 3442 Interfaces volume of IEEE Std 1003.1-2001 *scanf()* function to read the input file.

3443 The description of an individual record is as follows:

3444 "*<format>*", [*<arg1>*, *<arg2>*, . . . , *<argn>*]

3445 The *format* is a character string that contains three types of objects defined below:

- 3446 1. *Characters* that are not “escape sequences” or “conversion specifications”, as described
 3447 below, shall be copied to the output.
- 3448 2. *Escape Sequences* represent non-graphic characters.
- 3449 3. *Conversion Specifications* specify the output format of each argument; see below.

3450 The following characters have the following special meaning in the format string:

3451 ' ' (An empty character position.) Represents one or more *<blank>*s.

3452 Δ Represents exactly one *<space>*.

3453 Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

Table 5-1 Escape Sequences and Associated Actions

Escape Sequence	Represents Character	Terminal Action
'\\'	backslash	Print the character '\\ '.
'\a'	alert	Attempt to alert the user through audible or visible notification.
'\b'	backspace	Move the printing position to one column before the current position, unless the current position is the start of a line.
'\f'	form-feed	Move the printing position to the initial printing position of the next logical page.
'\n'	newline	Move the printing position to the start of the next line.
'\r'	carriage-return	Move the printing position to the start of the current line.
'\t'	tab	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
'\v'	vertical-tab	Move the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

Each conversion specification is introduced by the percent-sign character ('%'). After the character '%', the following shall appear in sequence:

flags Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

field width An optional string of decimal digits to specify a minimum field width. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left (or right, if the left-adjustment flag ('-'), described below, has been given) to the field width.

precision Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversion specifiers (the field is padded with leading zeros), the number of digits to appear after the radix character for the *e* and *f* conversion specifiers, the maximum number of significant digits for the *g* conversion specifier; or the maximum number of bytes to be written from a string in the *s* conversion specifier. The precision shall take the form of a period ('.') followed by a decimal digit string; a null digit string is treated as zero.

conversion specifier characters

A conversion specifier character (see below) that indicates the type of conversion to be applied.

The *flag* characters and their meanings are:

– The result of the conversion shall be left-justified within the field.

– The result of a signed conversion shall always begin with a sign ('+' or '-').

<space> If the first character of a signed conversion is not a sign, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.

The value shall be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s* conversion specifiers, the behavior is undefined. For the *o* conversion specifier, it shall increase the precision to force the first digit of the result to be a zero. For *x* or *X* conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively. For

3499		e, E, f, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. For g and G conversion specifiers, trailing zeros shall not be removed from the result as they usually are.
3500		
3501		
3502	0	For d, i, o, u, x, X, e, E, f, g, and G conversion specifiers, leading zeros (following any indication of sign or base) shall be used to pad to the field width; no space padding is performed. If the '0' and '-' flags both appear, the '0' flag shall be ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. For other conversion specifiers, the behavior is undefined.
3503		
3504		
3505		
3506		
3507		
3508		Each conversion specifier character shall result in fetching zero or more arguments. The results
3509		are undefined if there are insufficient arguments for the format. If the format is exhausted while
3510		arguments remain, the excess arguments shall be ignored.
3511		The conversion specifiers and their meanings are:
3512	d,i,o,u,x,X	The integer argument shall be written as signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and i specifiers shall convert to signed decimal in the style "[−] dddd". The x conversion specifier shall use the numbers and letters "0123456789abcdef" and the X conversion specifier shall use the numbers and letters "0123456789ABCDEF". The <i>precision</i> component of the argument shall specify the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting a zero value with a precision of 0 shall be no characters. If both the field width and precision are omitted, the implementation may precede, follow, or precede and follow numeric arguments of types d, i, and u with <blank>s; arguments of type o (octal) may be preceded with leading zeros.
3513		
3514		
3515		
3516		
3517		
3518		
3519		
3520		
3521		
3522		
3523		
3524		
3525	f	The floating-point number argument shall be written in decimal notation in the style "[−]ddd.ddd", where the number of digits after the radix character (shown here as a decimal point) shall be equal to the <i>precision</i> specification. The <i>LC_NUMERIC</i> locale category shall determine the radix character to use in this format. If the <i>precision</i> is omitted from the argument, six digits shall be written after the radix character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3526		
3527		
3528		
3529		
3530		
3531	e,E	The floating-point number argument shall be written in the style "[−]d.ddde±dd" (the symbol '±' indicates either a plus or minus sign), where there is one digit before the radix character (shown here as a decimal point) and the number of digits after it is equal to the precision. The <i>LC_NUMERIC</i> locale category shall determine the radix character to use in this format. When the precision is missing, six digits shall be written after the radix character; if the precision is 0, no radix character shall appear. The E conversion specifier shall produce a number with E instead of e introducing the exponent. The exponent shall always contain at least two digits. However, if the value to be written requires an exponent greater than two digits, additional exponent digits shall be written as necessary.
3532		
3533		
3534		
3535		
3536		
3537		
3538		
3539		
3540		
3541	g,G	The floating-point number argument shall be written in style f or e (or in style F or E in the case of a G conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style e (or E) shall be used only if the exponent resulting from the conversion is less than −4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.
3542		
3543		
3544		
3545		
3546		

3547 c The integer argument shall be converted to an **unsigned char** and the resulting
3548 byte shall be written.

3549 s The argument shall be taken to be a string and bytes from the string shall be
3550 written until the end of the string or the number of bytes indicated by the *precision*
3551 specification of the argument is reached. If the precision is omitted from the
3552 argument, it shall be taken to be infinite, so all bytes up to the end of the string
3553 shall be written.

3554 % Write a ' % ' character; no argument is converted.

3555 In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of
3556 a conversion is wider than the field width, the field is simply expanded to contain the conversion
3557 result. The term “field width” should not be confused with the term “precision” used in the
3558 description of %s.

3559 **Examples**

3560 To represent the output of a program that prints a date and time in the form Sunday, July 3,
3561 10:02, where *weekday* and *month* are strings:

3562 " %s, %s %d, %d: %.2d \n" <weekday>, <month>, <day>, <hour>, <min>

3563 To show ' π ' written to 5 decimal places:

3564 "pi = %.5f \n", <value of π >

3565 To show an input file format consisting of five colon-separated fields:

3566 " %s: %s: %s: %s: %s \n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>

Character Set

6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in Table 6-1. This is used to describe characters within the text of IEEE Std 1003.1-2001. The first eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the characters are defined in the ISO/IEC 10646-1:2000 standard.

Table 6-1 Portable Character Set

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(	<U0028>	LEFT PARENTHESIS
<right-parenthesis>	)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	–	<U002D>	HYPHEN-MINUS
<hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE

3608

3609

3610

3611

3612

3613

3614

3615

3616

3617

3618

3619

3620

3621

3622

3623

3624

3625

3626

3627

3628

3629

3630

3631

3632

3633

3634

3635

3636

3637

3638

3639

3640

3641

3642

3643

3644

3645

3646

3647

3648

3649

3650

3651

3652

3653

3654

3655

3656

Symbolic Name	Glyph	UCS	Description
<four>	4	<U0034>	DIGIT FOUR
<five>	5	<U0035>	DIGIT FIVE
<six>	6	<U0036>	DIGIT SIX
<seven>	7	<U0037>	DIGIT SEVEN
<eight>	8	<U0038>	DIGIT EIGHT
<nine>	9	<U0039>	DIGIT NINE
<colon>	:	<U003A>	COLON
<semicolon>	;	<U003B>	SEMICOLON
<less-than-sign>	<	<U003C>	LESS-THAN SIGN
<equals-sign>	=	<U003D>	EQUALS SIGN
<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
<question-mark>	?	<U003F>	QUESTION MARK
<commercial-at>	@	<U0040>	COMMERCIAL AT
<A>	A	<U0041>	LATIN CAPITAL LETTER A
	B	<U0042>	LATIN CAPITAL LETTER B
<C>	C	<U0043>	LATIN CAPITAL LETTER C
<D>	D	<U0044>	LATIN CAPITAL LETTER D
<E>	E	<U0045>	LATIN CAPITAL LETTER E
<F>	F	<U0046>	LATIN CAPITAL LETTER F
<G>	G	<U0047>	LATIN CAPITAL LETTER G
<H>	H	<U0048>	LATIN CAPITAL LETTER H
<I>	I	<U0049>	LATIN CAPITAL LETTER I
<J>	J	<U004A>	LATIN CAPITAL LETTER J
<K>	K	<U004B>	LATIN CAPITAL LETTER K
<L>	L	<U004C>	LATIN CAPITAL LETTER L
<M>	M	<U004D>	LATIN CAPITAL LETTER M
<N>	N	<U004E>	LATIN CAPITAL LETTER N
<O>	O	<U004F>	LATIN CAPITAL LETTER O
<P>	P	<U0050>	LATIN CAPITAL LETTER P
<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
<R>	R	<U0052>	LATIN CAPITAL LETTER R
<S>	S	<U0053>	LATIN CAPITAL LETTER S
<T>	T	<U0054>	LATIN CAPITAL LETTER T
<U>	U	<U0055>	LATIN CAPITAL LETTER U
<V>	V	<U0056>	LATIN CAPITAL LETTER V
<W>	W	<U0057>	LATIN CAPITAL LETTER W
<X>	X	<U0058>	LATIN CAPITAL LETTER X
<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
<left-square-bracket>	[	<U005B>	LEFT SQUARE BRACKET
<backslash>	\	<U005C>	REVERSE SOLIDUS
<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
<right-square-bracket>	]	<U005D>	RIGHT SQUARE BRACKET
<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
<low-line>	_	<U005F>	LOW LINE
<underscore>	_	<U005F>	LOW LINE

3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691

Symbolic Name	Glyph	UCS	Description
<grave-accent>	`	<U0060>	GRAVE ACCENT
<a>	a	<U0061>	LATIN SMALL LETTER A
	b	<U0062>	LATIN SMALL LETTER B
<c>	c	<U0063>	LATIN SMALL LETTER C
<d>	d	<U0064>	LATIN SMALL LETTER D
<e>	e	<U0065>	LATIN SMALL LETTER E
<f>	f	<U0066>	LATIN SMALL LETTER F
<g>	g	<U0067>	LATIN SMALL LETTER G
<h>	h	<U0068>	LATIN SMALL LETTER H
<i>	i	<U0069>	LATIN SMALL LETTER I
<j>	j	<U006A>	LATIN SMALL LETTER J
<k>	k	<U006B>	LATIN SMALL LETTER K
<l>	l	<U006C>	LATIN SMALL LETTER L
<m>	m	<U006D>	LATIN SMALL LETTER M
<n>	n	<U006E>	LATIN SMALL LETTER N
<o>	o	<U006F>	LATIN SMALL LETTER O
<p>	p	<U0070>	LATIN SMALL LETTER P
<q>	q	<U0071>	LATIN SMALL LETTER Q
<r>	r	<U0072>	LATIN SMALL LETTER R
<s>	s	<U0073>	LATIN SMALL LETTER S
<t>	t	<U0074>	LATIN SMALL LETTER T
<u>	u	<U0075>	LATIN SMALL LETTER U
<v>	v	<U0076>	LATIN SMALL LETTER V
<w>	w	<U0077>	LATIN SMALL LETTER W
<x>	x	<U0078>	LATIN SMALL LETTER X
<y>	y	<U0079>	LATIN SMALL LETTER Y
<z>	z	<U007A>	LATIN SMALL LETTER Z
<left-brace>	{	<U007B>	LEFT CURLY BRACKET
<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
<vertical-line>		<U007C>	VERTICAL LINE
<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
<tilde>	~	<U007E>	TILDE

3692
3693
3694

IEEE Std 1003.1-2001 uses character names other than the above, but only in an informative way; for example, in examples to illustrate the use of characters beyond the portable character set with the facilities of IEEE Std 1003.1-2001.

3695
3696
3697
3698
3699

Table 6-1 (on page 115) defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. The table contains more than one symbolic character name for characters whose traditional name differs from the chosen name. Characters defined in Table 6-2 (on page 120) may also be used in character set description files.

3700
3701

IEEE Std 1003.1-2001 places only the following requirements on the encoded values of the characters in the portable character set:

3702
3703
3704
3705
3706

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, if an application accesses any pair of locales where the character encodings differ, or accesses data from an application running in a locale which has different encodings from the application's current locale, the results are unspecified.

- 3707 • The encoded values associated with the digits 0 to 9 shall be such that the value of each
- 3708 character after 0 shall be one greater than the value of the previous character.
- 3709 • A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- 3710 • The encoded values associated with the members of the portable character set are each
- 3711 represented in a single byte. Moreover, if the value is stored in an object of C-language type
- 3712 **char**, it is guaranteed to be positive (except the NUL, which is always zero).
- 3713 Conforming implementations shall support certain character and character set attributes, as
- 3714 defined in Section 7.2 (on page 124).

3715 6.2 Character Encoding

3716 The POSIX locale contains the characters in Table 6-1 (on page 115), which have the properties

3717 listed in Section 7.3.1 (on page 126). In other locales, the presence, meaning, and representation

3718 of any additional characters are locale-specific.

3719 In locales other than the POSIX locale, a character may have a state-dependent encoding. There

3720 are two types of these encodings:

- 3721 • A single-shift encoding (where each character not in the initial shift state is preceded by a
- 3722 shift code) can be defined if each shift-code and character sequence is considered a multi-
- 3723 byte character. This is done using the concatenated-constant format in a character set
- 3724 description file, as described in Section 6.4 (on page 119). If the implementation supports a
- 3725 character encoding of this type, all of the standard utilities in the Shell and Utilities volume of
- 3726 IEEE Std 1003.1-2001 shall support it. Use of a single-shift encoding with any of the functions
- 3727 in the System Interfaces volume of IEEE Std 1003.1-2001 that do not specifically mention the
- 3728 effects of state-dependent encoding is implementation-defined.
- 3729 • A locking-shift encoding (where the state of the character is determined by a shift code that
- 3730 may affect more than the single character following it) cannot be defined with the current
- 3731 character set description file format. Use of a locking-shift encoding with any of the standard
- 3732 utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 or with any of the functions
- 3733 in the System Interfaces volume of IEEE Std 1003.1-2001 that do not specifically mention the
- 3734 effects of state-dependent encoding is implementation-defined.

3735 While in the initial shift state, all characters in the portable character set shall retain their usual

3736 interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the

3737 sequence shall be a function of the current shift state. A byte with all bits zero shall be

3738 interpreted as the null character independent of shift state. Thus a byte with all bits zero shall

3739 never occur in the second or subsequent bytes of a character.

3740 The maximum allowable number of bytes in a character in the current locale shall be indicated

3741 by {MB_CUR_MAX}, defined in the **<stdlib.h>** header and by the **<mb_cur_max>** value in a

3742 character set description file; see Section 6.4 (on page 119). The implementation's maximum

3743 number of bytes in a character shall be defined by the C-language macro {MB_LEN_MAX}.

6.3 C Language Wide-Character Codes

In the shell, the standard utilities are written so that the encodings of characters are described by the locale's *LC_CTYPE* definition (see Section 7.3.1 (on page 126)) and there is no differentiation between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C language, a differentiation is made. To ease the handling of variable length characters, the C language has introduced the concept of wide-character codes.

All wide-character codes in a given process consist of an equal number of bits. This is in contrast to characters, which can consist of a variable number of bytes. The byte or byte sequence that represents a character can also be represented as a wide-character code. Wide-character codes thus provide a uniform size for manipulating text data. A wide-character code having all bits zero is the null wide-character code (see Section 3.246 (on page 69)), and terminates wide-character strings (see Section 3.432 (on page 95)). The wide-character value for each member of the portable character set shall equal its value when used as the lone character in an integer character constant. Wide-character codes for other characters are locale and implementation-defined. State shift bytes shall not have a wide-character code representation.

6.4 Character Set Description File

Implementations shall provide a character set description file for at least one coded character set supported by the implementation. These files are referred to elsewhere in IEEE Std 1003.1-2001 as *charmap* files. It is implementation-defined whether or not users or applications can provide additional character set description files.

IEEE Std 1003.1-2001 does not require that multiple character sets or codesets be supported. Although multiple *charmap* files are supported, it is the responsibility of the implementation to provide the file or files; if only one is provided, only that one is accessible using the *localedef* utility's *-f* option.

Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard position values as the encoding values, shall define characteristics for the coded character set and the encoding for the characters specified in Table 6-1 (on page 115), and may define encoding for additional characters supported by the implementation. Other information about the coded character set may also be in the file. Coded character set character values shall be defined using symbolic character names followed by character encoding values.

Each symbolic name specified in Table 6-1 (on page 115) shall be included in the file and shall be mapped to a unique coding value, except as noted below. The glyphs '{', '}', '_', '-', '/', '\', '.', and '^' have more than one symbolic name; all symbolic names for each such glyph shall be included, each with identical encoding. If some or all of the control characters identified in Table 6-2 (on page 120) are supported by the implementation, the symbolic names and their corresponding encoding values shall be included in the file. Some of the encodings associated with the symbolic names in Table 6-2 (on page 120) may be the same as characters found in Table 6-1 (on page 115); both names shall be provided for each encoding.

Table 6-2 Control Character Set

<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
<CAN>	<DEL>	<ETB>	<IS1>	<RS>	<SYN>
<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
<DC1>		<FF>	<IS3>	<SO>	<VT>

The following declarations can precede the character definitions. Each shall consist of the symbol shown in the following list, starting in column 1, including the surrounding brackets, followed by one or more <blank>s, followed by the value to be assigned to the symbol.

<code_set_name> The name of the coded character set for which the character set description file is defined. The characters of the name shall be taken from the set of characters with visible glyphs defined in Table 6-1 (on page 115).

<mb_cur_max> The maximum number of bytes in a multi-byte character. This shall default to 1.

<mb_cur_min> An unsigned positive integer value that defines the minimum number of bytes in a character for the encoded character set. On XSI-conformant systems, <mb_cur_min> shall always be 1.

<escape_char> The character used to indicate that the characters following shall be interpreted in a special way, as defined later in this section. This shall default to backslash (' \ '), which is the character used in all the following text and examples, unless otherwise noted.

<comment_char> The character that, when placed in column 1 of a charmap line, is used to indicate that the line shall be ignored. The default character shall be the number sign (' # ').

The character set mapping definitions shall be all the lines immediately following an identifier line containing the string "CHARMAP" starting in column 1, and preceding a trailer line containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a <comment_char> in the first column shall be ignored. Each non-comment line of the character set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file) shall be in either of two forms:

```
"%s %s %s\n", <symbolic-name>, <encoding>, <comments>
```

or:

```
"%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,  
    <encoding>, <comments>
```

In the first format, the line in the character set mapping definition shall define a single symbolic name and a corresponding encoding. A symbolic name is one or more characters from the set shown with visible glyphs in Table 6-1 (on page 115), enclosed between angle brackets. A character following an escape character is interpreted as itself; for example, the sequence "<\\>>" represents the symbolic name "<\\>" enclosed between angle brackets.

In the second format, the line in the character set mapping definition shall define a range of one or more symbolic names. In this form, the symbolic names shall consist of zero or more non-numeric characters from the set shown with visible glyphs in Table 6-1 (on page 115), followed by an integer formed by one or more decimal digits. Both integers shall contain the same number of digits. The characters preceding the integer shall be identical in the two symbolic names, and

the integer formed by the digits in the second symbolic name shall be equal to or greater than the integer formed by the digits in the first name. This shall be interpreted as a series of symbolic names formed from the common part and each of the integers between the first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

A character set mapping definition line shall exist for all symbolic names specified in Table 6-1 (on page 115), and shall define the coded character value that corresponds to the character indicated in the table, or the coded character value that corresponds to the control character symbolic name. If the control characters commonly associated with the symbolic names in Table 6-2 (on page 120) are supported by the implementation, the symbolic name and the corresponding encoding value shall be included in the file. Additional unique symbolic names may be included. A coded character value can be represented by more than one symbolic name.

The encoding part is expressed as one (for single-byte character values) or more concatenated decimal, octal, or hexadecimal constants in the following formats:

```
%cd%u", <escape_char>, <decimal byte value>
%cx%x", <escape_char>, <hexadecimal byte value>
%c%o", <escape_char>, <octal byte value>
```

Decimal constants shall be represented by two or three decimal digits, preceded by the escape character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143". Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal constants shall be represented by two or three octal digits, preceded by the escape character; for example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an 8-bit byte. When constants are concatenated for multi-byte character values, they shall be of the same type, and interpreted in byte order from first to last with the least significant byte of the multi-byte character specified by the last constant. The manner in which these constants are represented in the character stored in the system is implementation-defined. (This notation was chosen for reasons of portability. There is no requirement that the internal representation in the computer memory be in this same order.) Omitting bytes from a multi-byte character definition produces undefined results.

In lines defining ranges of symbolic names, the encoded value shall be the value for the first symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names defined by the range shall have encoding values in increasing order. Bytes shall be treated as unsigned octets, and carry shall be propagated between the bytes as necessary to represent the range. For example, the line:

```
<j0101>...<j0104>  \d129\d254
```

is interpreted as:

```
<j0101>          \d129\d254
<j0102>          \d129\d255
<j0103>          \d130\d0
<j0104>          \d130\d1
```

The comment is optional.

The following declarations can follow the character set mapping definitions (after the "END CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in column 1, followed by the value(s) to be associated to the keyword, as defined below.

WIDTH An unsigned positive integer value defining the column width (see Section 3.103 (on page 49)) for the printable characters in the coded character set specified in

Table 6-1 (on page 115) and Table 6-2 (on page 120). Coded character set character values shall be defined using symbolic character names followed by column width values. Defining a character with more than one **WIDTH** produces undefined results. The **END WIDTH** keyword shall be used to terminate the **WIDTH** definitions. Specifying the width of a non-printable character in a **WIDTH** declaration produces undefined results.

WIDTH_DEFAULT

An unsigned positive integer value defining the default column width for any printable character not listed by one of the **WIDTH** keywords. If no **WIDTH_DEFAULT** keyword is included in the charmap, the default character width shall be 1.

Example

After the "END CHARMAP" statement, a syntax for a width definition would be:

```
WIDTH
<A> 1
<B> 1
<C>...<Z> 1
...
<foo1>...<foon> 2
...
END WIDTH
```

In this example, the numerical code point values represented by the symbols **<A>** and **** are assigned a width of 1. The code point values **<C>** to **<Z>** inclusive (**<C>**, **<D>**, **<E>**, and so on) are also assigned a width of 1. Using **<A>...<Z>** would have required fewer lines, but the alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have been added as appropriate.

6.4.1 State-Dependent Character Encodings

This section addresses the use of state-dependent character encodings (that is, those in which the encoding of a character is dependent on one or more shift codes that may precede it).

A single-shift encoding (where each character not in the initial shift state is preceded by a shift code) can be defined in the charmap format if each shift-code/character sequence is considered a multi-byte character, defined using the concatenated-constant format described in Section 6.4 (on page 119). If the implementation supports a character encoding of this type, all of the standard utilities shall support it. A locking-shift encoding (where the state of the character is determined by a shift code that may affect more than the single character following it) could be defined with an extension to the charmap format described in Section 6.4 (on page 119). If the implementation supports a character encoding of this type, any of the standard utilities that describe character (*versus* byte) or text-file manipulation shall have the following characteristics:

1. The utility shall process the statefully encoded data as a concatenation of state-independent characters. The presence of redundant locking shifts shall not affect the comparison of two statefully encoded strings.
2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall produce output that contains locking shifts at the beginning or end of the resulting data, if appropriate, to retain correct state information.

3919

7.1 General

A locale is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

LC_CTYPE Character classification and case conversion.

LC_COLLATE Collation order.

LC_MONETARY Monetary formatting.

LC_NUMERIC Numeric, non-monetary formatting.

LC_TIME Date and time formats.

LC_MESSAGES Formats of informative and diagnostic messages and interactive responses.

The standard utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 shall base their behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility. The behavior of some of the C-language functions defined in the System Interfaces volume of IEEE Std 1003.1-2001 shall also be modified based on the current locale, as defined by the last call to *setlocale()*.

Locales other than those supplied by the implementation can be created via the *localedef* utility, provided that the *_POSIX2_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not provided, all implementations conforming to the System Interfaces volume of IEEE Std 1003.1-2001 shall provide one or more locales that behave as described in this chapter. The input to the utility is described in Section 7.3 (on page 124). The value that is used to specify a locale when using environment variables shall be the string specified as the *name* operand to the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for the POSIX locale (see Section 7.2 (on page 124)). When the value of a locale environment variable begins with a slash ('/'), it shall be interpreted as the pathname of the locale definition; the type of file (regular, directory, and so on) used to store the locale definition is implementation-defined. If the value does not begin with a slash, the mechanism used to locate the locale is implementation-defined.

If different character sets are used by the locale categories, the results achieved by an application utilizing these categories are undefined. Likewise, if different codesets are used for the data being processed by interfaces whose behavior is dependent on the current locale, or the codeset is different from the codeset assumed when the locale was created, the result is also undefined.

Applications can select the desired locale by invoking the *setlocale()* function (or equivalent) with the appropriate value. If the function is invoked with an empty string, such as:

```
setlocale(LC_ALL, "");
```

the value of the corresponding environment variable is used. If the environment variable is unset or is set to the empty string, the implementation shall set the appropriate environment as defined in Chapter 8 (on page 161).

7.2 POSIX Locale

Conforming systems shall provide a POSIX locale, also known as the C locale. The behavior of standard utilities and functions in the POSIX locale shall be as if the locale was defined via the *localedef* utility with input data from the POSIX locale tables in Section 7.3.

The tables in Section 7.3 describe the characteristics and behavior of the POSIX locale for data consisting entirely of characters from the portable character set and the control character set. For other characters, the behavior is unspecified. For C-language programs, the POSIX locale shall be the default locale when the *setlocale()* function is not called.

The POSIX locale can be specified by assigning to the appropriate environment variables the values "C" or "POSIX".

All implementations shall define a locale as the default locale, to be invoked when no environment variables are set, or set to the empty string. This default locale can be the POSIX locale or any other implementation-defined locale. Some implementations may provide facilities for local installation administrators to set the default locale, customizing it for each location. IEEE Std 1003.1-2001 does not require such a facility.

7.3 Locale Definition

The capability to specify additional locales to those provided by an implementation is optional, denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only implementation-supplied locales are available. Such locales shall be documented using the format specified in this section.

Locales can be described with the file format presented in this section. The file format is that accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the "locale definition file", but no locales shall be affected by this file unless it is processed by *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility shall apply to *localedef* or to any other similar utility used to install locale information using the locale definition file format described here.

The locale definition file shall contain one or more locale category source definitions, and shall not contain more than one definition for the same locale category. If the file contains source definitions for more than one category, implementation-defined categories, if present, shall appear after the categories defined by Section 7.1 (on page 123). A category source definition contains either the definition of a category or a **copy** directive. For a description of the **copy** directive, see *localedef*. In the event that some of the information for a locale category, as specified in this volume of IEEE Std 1003.1-2001, is missing from the locale source definition, the behavior of that category, if it is referenced, is unspecified.

A category source definition shall consist of a category header, a category body, and a category trailer. A category header shall consist of the character string naming of the category, beginning with the characters `LC_`. The category trailer shall consist of the string "END", followed by one or more <blank>s and the string used in the corresponding category header.

The category body shall consist of one or more lines of text. Each line shall contain an identifier, optionally followed by one or more operands. Identifiers shall be either keywords, identifying a particular locale element, or collating elements. In addition to the keywords defined in this volume of IEEE Std 1003.1-2001, the source can contain implementation-defined keywords. Each keyword within a locale shall have a unique name (that is, two categories cannot have a commonly-named keyword); no keyword shall start with the characters `LC_`. Identifiers shall be separated from the operands by one or more <blank>s.

Operands shall be characters, collating elements, or strings of characters. Strings shall be enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the *<escape character>*, described below. When a keyword is followed by more than one operand, the operands shall be separated by semicolons; *<blank>*s shall be allowed both before and after a semicolon.

The first category header in the file can be preceded by a line modifying the comment character. It shall have the following format, starting in column 1:

```
"comment_char %c\n", <comment character>
```

The comment character shall default to the number sign ('#'). Blank lines and lines containing the *<comment character>* in the first position shall be ignored.

The first category header in the file can be preceded by a line modifying the escape character to be used in the file. It shall have the following format, starting in column 1:

```
"escape_char %c\n", <escape character>
```

The escape character shall default to backslash, which is the character used in all examples shown in this volume of IEEE Std 1003.1-2001.

A line can be continued by placing an escape character as the last character on the line; this continuation character shall be discarded from the input. Although the implementation need not accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall place no limits on the accumulated length of the continued line. Comment lines shall not be continued on a subsequent line using an escaped *<newline>*.

Individual characters, characters in strings, and collating elements shall be represented using symbolic names, as defined below. In addition, characters can be represented using the characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic notation is used, the resultant locale definitions are in many cases not portable between systems. The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when used to represent itself it shall be preceded by the escape character. The following rules apply to character representation:

1. A character can be represented via a symbolic name, enclosed within angle brackets '<' and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic name defined in the charmap file specified via the *localedef -f* option, and it shall be replaced by a character value determined from the value associated with the symbolic name in the charmap file. The use of a symbolic name not found in the charmap file shall constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it shall constitute a warning condition (see *localedef* for a description of actions resulting from errors and warnings). The specification of a symbolic name in a **collating-element** or **collating-symbol** section that duplicates a symbolic name in the charmap file (if present) shall be an error. Use of the escape character or a right angle bracket within a symbolic name is invalid unless the character is preceded by the escape character.

For example:

```
<c>;<c-cedilla> "<M><a><y>"
```

2. A character in the portable character set can be represented by the character itself, in which case the value of the character is implementation-defined. (Implementations may allow other characters to be represented as themselves, but such locale definitions are not portable.) Within a string, the double-quote character, the escape character, and the right angle bracket character shall be escaped (preceded by the escape character) to be interpreted as the character itself. Outside strings, the characters:

4049 , ; < > escape_char

4050 shall be escaped to be interpreted as the character itself.

4051 For example:

4052 c "May"

4053 3. A character can be represented as an octal constant. An octal constant shall be specified as
4054 the escape character followed by two or three octal digits. Each constant shall represent a
4055 byte value. Multi-byte values can be represented by concatenated constants specified in
4056 byte order with the last constant specifying the least significant byte of the character.

4057 For example:

4058 \143;\347;\143\150 "\115\141\171"

4059 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be
4060 specified as the escape character followed by an 'x' followed by two hexadecimal digits.
4061 Each constant shall represent a byte value. Multi-byte values can be represented by
4062 concatenated constants specified in byte order with the last constant specifying the least
4063 significant byte of the character.

4064 For example:

4065 \x63;\xe7;\x63\x68 "\x4d\x61\x79"

4066 5. A character can be represented as a decimal constant. A decimal constant shall be specified
4067 as the escape character followed by a 'd' followed by two or three decimal digits. Each
4068 constant represents a byte value. Multi-byte values can be represented by concatenated
4069 constants specified in byte order with the last constant specifying the least significant byte
4070 of the character.

4071 For example:

4072 \d99;\d231;\d99\d104 "\d77\d97\d121"

4073 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the
4074 escape character. Only characters existing in the character set for which the locale definition is
4075 created shall be specified, whether using symbolic names, the characters themselves, or octal,
4076 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the
4077 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not
4078 present in the charmap file can be specified and shall be ignored, as specified under item 1
4079 above.

4080 7.3.1 LC_CTYPE

4081 The *LC_CTYPE* category shall define character classification, case conversion, and other
4082 character attributes. In addition, a series of characters can be represented by three adjacent
4083 periods representing an ellipsis symbol ("..."). The ellipsis specification shall be interpreted as
4084 meaning that all values between the values preceding and following it represent valid
4085 characters. The ellipsis specification shall be valid only within a single encoded character set;
4086 that is, within a group of characters of the same size. An ellipsis shall be interpreted as including
4087 in the list all characters with an encoded value higher than the encoded value of the character
4088 preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

4089 For example:

4090 \x30;...;\x39;

includes in the character class all characters with encoded values between the endpoints.

The following keywords shall be recognized. In the descriptions, the term “automatically included” means that it shall not be an error either to include or omit any of the referenced characters; the implementation provides them if missing (even if the entire keyword is missing) and accepts them silently if present. When the implementation automatically includes a missing character, it shall have an encoded value dependent on the charmap file in effect (see the description of the *localedef* **-f** option); otherwise, it shall have a value derived from an implementation-defined character mapping.

The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included characters. These only need to be specified if the character values (that is, encoding) differ from the implementation default values. It is not possible to define a locale without these automatically included characters unless some implementation extension is used to prevent their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and, thus, it might not be possible for conforming applications to work properly.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

upper Define characters to be classified as uppercase letters.

In the POSIX locale, the 26 uppercase letters shall be included:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as defined in Section 6.4 (on page 119) (the portable character set), are automatically included in this class.

lower Define characters to be classified as lowercase letters.

In the POSIX locale, the 26 lowercase letters shall be included:

a b c d e f g h i j k l m n o p q r s t u v w x y z

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the portable character set are automatically included in this class.

alpha Define characters to be classified as letters.

In the POSIX locale, all characters in the classes **upper** and **lower** shall be included.

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. Characters classified as either **upper** or **lower** are automatically included in this class.

digit Define the characters to be classified as numeric digits.

In the POSIX locale, only:

0 1 2 3 4 5 6 7 8 9

shall be included.

In a locale definition file, only the digits <zero>, <one>, <two>, <three>, <four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in contiguous ascending sequence by numerical value. The digits <zero> to <nine> of the portable character set are automatically included in this class.

4134	alnum	Define characters to be classified as letters and numeric digits. Only the
4135		characters specified for the alpha and digit keywords shall be specified.
4136		Characters specified for the keywords alpha and digit are automatically
4137		included in this class.
4138	space	Define characters to be classified as white-space characters.
4139		In the POSIX locale, at a minimum, the <space>, <form-feed>, <newline>,
4140		<carriage-return>, <tab>, and <vertical-tab> shall be included.
4141		In a locale definition file, no character specified for the keywords upper ,
4142		lower , alpha , digit , graph , or xdigit shall be specified. The <space>, <form-
4143		feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable
4144		character set, and any characters included in the class blank are automatically
4145		included in this class.
4146	cntrl	Define characters to be classified as control characters.
4147		In the POSIX locale, no characters in classes alpha or print shall be included.
4148		In a locale definition file, no character specified for the keywords upper ,
4149		lower , alpha , digit , punct , graph , print , or xdigit shall be specified.
4150	punct	Define characters to be classified as punctuation characters.
4151		In the POSIX locale, neither the <space> nor any characters in classes alpha ,
4152		digit , or cntrl shall be included.
4153		In a locale definition file, no character specified for the keywords upper ,
4154		lower , alpha , digit , cntrl , xdigit , or as the <space> shall be specified.
4155	graph	Define characters to be classified as printable characters, not including the
4156		<space>.
4157		In the POSIX locale, all characters in classes alpha , digit , and punct shall be
4158		included; no characters in class cntrl shall be included.
4159		In a locale definition file, characters specified for the keywords upper , lower ,
4160		alpha , digit , xdigit , and punct are automatically included in this class. No
4161		character specified for the keyword cntrl shall be specified.
4162	print	Define characters to be classified as printable characters, including the
4163		<space>.
4164		In the POSIX locale, all characters in class graph shall be included; no
4165		characters in class cntrl shall be included.
4166		In a locale definition file, characters specified for the keywords upper , lower ,
4167		alpha , digit , xdigit , punct , graph , and the <space> are automatically included
4168		in this class. No character specified for the keyword cntrl shall be specified.
4169	xdigit	Define the characters to be classified as hexadecimal digits.
4170		In the POSIX locale, only:
4171		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4172		shall be included.
4173		In a locale definition file, only the characters defined for the class digit shall be
4174		specified, in contiguous ascending sequence by numerical value, followed by
4175		one or more sets of six characters representing the hexadecimal digits 10 to 15

4176		inclusive, with each set in ascending order (for example, <A>, , <C>, <D>, <E>, <F>, <a>, , <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the
4177		uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the
4178		portable character set are automatically included in this class.
4179		
4180	blank	Define characters to be classified as <blank>s.
4181		In the POSIX locale, only the <space> and <tab> shall be included.
4182		In a locale definition file, the <space> and <tab> are automatically included in
4183		this class.
4184	charclass	Define one or more locale-specific character class names as strings separated
4185		by semicolons. Each named character class can then be defined subsequently
4186		in the <i>LC_CTYPE</i> definition. A character class name shall consist of at least
4187		one and at most {CHARCLASS_NAME_MAX} bytes of alphanumeric
4188		characters from the portable filename character set. The first character of a
4189		character class name shall not be a digit. The name shall not match any of the
4190		<i>LC_CTYPE</i> keywords defined in this volume of IEEE Std 1003.1-2001. Future
4191		revisions of IEEE Std 1003.1-2001 will not specify any <i>LC_CTYPE</i> keywords
4192		containing uppercase letters.
4193	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific
4194		character class. In the POSIX locale, locale-specific named character classes
4195		need not exist.
4196		If a class name is defined by a charclass keyword, but no characters are
4197		subsequently assigned to it, this is not an error; it represents a class without
4198		any characters belonging to it.
4199		The <i>charclass-name</i> can be used as the <i>property</i> argument to the <i>wctype()</i>
4200		function, in regular expression and shell pattern-matching bracket
4201		expressions, and by the <i>tr</i> command.
4202	toupper	Define the mapping of lowercase letters to uppercase letters.
4203		In the POSIX locale, at a minimum, the 26 lowercase characters:
4204		a b c d e f g h i j k l m n o p q r s t u v w x y z
4205		shall be mapped to the corresponding 26 uppercase characters:
4206		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4207		In a locale definition file, the operand shall consist of character pairs,
4208		separated by semicolons. The characters in each character pair shall be
4209		separated by a comma and the pair enclosed by parentheses. The first
4210		character in each pair is the lowercase letter, the second the corresponding
4211		uppercase letter. Only characters specified for the keywords lower and upper
4212		shall be specified. The lowercase letters <a> to <z>, and their corresponding
4213		uppercase letters <A> to <Z>, of the portable character set are automatically
4214		included in this mapping, but only when the toupper keyword is omitted
4215		from the locale definition.
4216	tolower	Define the mapping of uppercase letters to lowercase letters.
4217		In the POSIX locale, at a minimum, the 26 uppercase characters:
4218		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

shall be mapped to the corresponding 26 lowercase characters:

a b c d e f g h i j k l m n o p q r s t u v w x y z

In a locale definition file, the operand shall consist of character pairs, separated by semicolons. The characters in each character pair shall be separated by a comma and the pair enclosed by parentheses. The first character in each pair is the uppercase letter, the second the corresponding lowercase letter. Only characters specified for the keywords **lower** and **upper** shall be specified. If the **tolower** keyword is omitted from the locale definition, the mapping is the reverse mapping of the one specified for **toupper**.

The following table shows the character class combinations allowed:

Table 7-1 Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
upper	—	—	A	x	x	x	x	A	A	—	x
lower	—	—	A	x	x	x	x	A	A	—	x
alpha	—	—		x	x	x	x	A	A	—	x
digit	x	x	x		x	x	x	A	A	A	x
space	x	x	x	x		—	*	*	*	x	—
cntrl	x	x	x	x	—		x	x	x	x	—
punct	x	x	x	x	—	x		A	A	x	—
graph	—	—	—	—	—	x	—		A	—	—
print	—	—	—	—	—	x	—	—		—	—
xdigit	—	—	—	—	x	x	x	A	A		x
blank	x	x	x	x	A	—	*	*	*	x	

Notes:

1. Explanation of codes:

A Automatically included; see text.

— Permitted.

x Mutually-exclusive.

* See note 2.

2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters can be classified as any of **punct**, **graph**, or **print**.

7.3.1.1 LC_CTYPE Category in the POSIX Locale

The character classifications for the POSIX locale follow; the code listing depicts the *localedef* input, and the table represents the same information, sorted by character.

```
LC_CTYPE
# The following is the POSIX locale LC_CTYPE.
# "alpha" is by default "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by default "alnum", "punct", and the <space>
# "graph" is by default "alnum" and "punct"
#
upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
```



```

4263      <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4264      #
4265      lower  <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4266             <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4267      #
4268      digit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4269             <seven>;<eight>;<nine>
4270      #
4271      space  <tab>;<newline>;<vertical-tab>;<form-feed>;\
4272             <carriage-return>;<space>
4273      #
4274      cntrl  <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4275             <form-feed>;<carriage-return>;\
4276             <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4277             <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4278             <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4279             <IS1>;<DEL>
4280      #
4281      punct  <exclamation-mark>;<quotation-mark>;<number-sign>;\
4282             <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4283             <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4284             <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4285             <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4286             <greater-than-sign>;<question-mark>;<commercial-at>;\
4287             <left-square-bracket>;<backslash>;<right-square-bracket>;\
4288             <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4289             <vertical-line>;<right-curly-bracket>;<tilde>
4290      #
4291      xdigit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4292             <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4293      #
4294      blank  <space>;<tab>
4295      #
4296      toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
4297             (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
4298             (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
4299             (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
4300             (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
4301      #
4302      tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
4303             (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
4304             (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
4305             (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
4306             (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
4307      END LC_CTYPE

```

	Symbolic Name	Other Case	Character Classes
4308	<NUL>		cntrl
4309	<SOH>		cntrl
4310	<STX>		cntrl
4311	<ETX>		cntrl
4312	<EOT>		cntrl
4313	<ENQ>		cntrl
4314	<ACK>		cntrl
4315	<alert>		cntrl
4316	<backspace>		cntrl
4317	<tab>		cntrl, space, blank
4318	<newline>		cntrl, space
4319	<vertical-tab>		cntrl, space
4320	<form-feed>		cntrl, space
4321	<carriage-return>		cntrl, space
4322	<SO>		cntrl
4323	<SI>		cntrl
4324	<DLE>		cntrl
4325	<DC1>		cntrl
4326	<DC2>		cntrl
4327	<DC3>		cntrl
4328	<DC4>		cntrl
4329	<NAK>		cntrl
4330	<SYN>		cntrl
4331	<ETB>		cntrl
4332	<CAN>		cntrl
4333			cntrl
4334	<SUB>		cntrl
4335	<ESC>		cntrl
4336	<IS4>		cntrl
4337	<IS3>		cntrl
4338	<IS2>		cntrl
4339	<IS1>		cntrl
4340	<space>		space, print, blank
4341	<exclamation-mark>		punct, print, graph
4342	<quotation-mark>		punct, print, graph
4343	<number-sign>		punct, print, graph
4344	<dollar-sign>		punct, print, graph
4345	<percent-sign>		punct, print, graph
4346	<ampersand>		punct, print, graph
4347	<apostrophe>		punct, print, graph
4348	<left-parenthesis>		punct, print, graph
4349	<right-parenthesis>		punct, print, graph
4350	<asterisk>		punct, print, graph
4351	<plus-sign>		punct, print, graph
4352	<comma>		punct, print, graph
4353	<hyphen>		punct, print, graph
4354	<period>		punct, print, graph

	Symbolic Name	Other Case	Character Classes
4357	<slash>		punct, print, graph
4358	<zero>		digit, xdigit, print, graph
4359	<one>		digit, xdigit, print, graph
4360	<two>		digit, xdigit, print, graph
4361	<three>		digit, xdigit, print, graph
4362	<four>		digit, xdigit, print, graph
4363	<five>		digit, xdigit, print, graph
4364	<six>		digit, xdigit, print, graph
4365	<seven>		digit, xdigit, print, graph
4366	<eight>		digit, xdigit, print, graph
4367	<nine>		digit, xdigit, print, graph
4368	<colon>		punct, print, graph
4369	<semicolon>		punct, print, graph
4370	<less-than-sign>		punct, print, graph
4371	<equals-sign>		punct, print, graph
4372	<greater-than-sign>		punct, print, graph
4373	<question-mark>		punct, print, graph
4374	<commercial-at>		punct, print, graph
4375	<A>	<a>	upper, xdigit, alpha, print, graph
4376			upper, xdigit, alpha, print, graph
4377	<C>	<c>	upper, xdigit, alpha, print, graph
4378	<D>	<d>	upper, xdigit, alpha, print, graph
4379	<E>	<e>	upper, xdigit, alpha, print, graph
4380	<F>	<f>	upper, xdigit, alpha, print, graph
4381	<G>	<g>	upper, alpha, print, graph
4382	<H>	<h>	upper, alpha, print, graph
4383	<I>	<i>	upper, alpha, print, graph
4384	<J>	<j>	upper, alpha, print, graph
4385	<K>	<k>	upper, alpha, print, graph
4386	<L>	<l>	upper, alpha, print, graph
4387	<M>	<m>	upper, alpha, print, graph
4388	<N>	<n>	upper, alpha, print, graph
4389	<O>	<o>	upper, alpha, print, graph
4390	<P>	<p>	upper, alpha, print, graph
4391	<Q>	<q>	upper, alpha, print, graph
4392	<R>	<r>	upper, alpha, print, graph
4393	<S>	<s>	upper, alpha, print, graph
4394	<T>	<t>	upper, alpha, print, graph
4395	<U>	<u>	upper, alpha, print, graph
4396	<V>	<v>	upper, alpha, print, graph
4397	<W>	<w>	upper, alpha, print, graph
4400	<X>	<x>	upper, alpha, print, graph
4401	<Y>	<y>	upper, alpha, print, graph
4402	<Z>	<z>	upper, alpha, print, graph
4403	<left-square-bracket>		punct, print, graph
4404	<backslash>		punct, print, graph
4405	<right-square-bracket>		punct, print, graph

Symbolic Name	Other Case	Character Classes
<circumflex>		punct, print, graph
<underscore>		punct, print, graph
<grave-accent>		punct, print, graph
<a>	<A>	lower, xdigit, alpha, print, graph
		lower, xdigit, alpha, print, graph
<c>	<C>	lower, xdigit, alpha, print, graph
<d>	<D>	lower, xdigit, alpha, print, graph
<e>	<E>	lower, xdigit, alpha, print, graph
<f>	<F>	lower, xdigit, alpha, print, graph
<g>	<G>	lower, alpha, print, graph
<h>	<H>	lower, alpha, print, graph
<i>	<I>	lower, alpha, print, graph
<j>	<J>	lower, alpha, print, graph
<k>	<K>	lower, alpha, print, graph
<l>	<L>	lower, alpha, print, graph
<m>	<M>	lower, alpha, print, graph
<n>	<N>	lower, alpha, print, graph
<o>	<O>	lower, alpha, print, graph
<p>	<P>	lower, alpha, print, graph
<q>	<Q>	lower, alpha, print, graph
<r>	<R>	lower, alpha, print, graph
<s>	<S>	lower, alpha, print, graph
<t>	<T>	lower, alpha, print, graph
<u>	<U>	lower, alpha, print, graph
<v>	<V>	lower, alpha, print, graph
<w>	<W>	lower, alpha, print, graph
<x>	<X>	lower, alpha, print, graph
<y>	<Y>	lower, alpha, print, graph
<z>	<Z>	lower, alpha, print, graph
<left-curly-bracket>		punct, print, graph
<vertical-line>		punct, print, graph
<right-curly-bracket>		punct, print, graph
<tilde>		punct, print, graph
<DEL>		cntrl

7.3.2 LC_COLLATE

The *LC_COLLATE* category provides a collation sequence definition for numerous utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 (*sort*, *uniq*, and so on), regular expression matching (see Chapter 9 (on page 169)), and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()* functions in the System Interfaces volume of IEEE Std 1003.1-2001.

A collation sequence definition shall define the relative order between collating elements (characters and multi-character collating elements) in the locale. This order is expressed in terms of collation values; that is, by assigning each element one or more collation values (also known as collation weights). This does not imply that implementations shall assign such values, but that ordering of strings using the resultant collation definition in the locale behaves as if such assignment is done and used in the collation process. At least the following capabilities are provided:

1. **Multi-character collating elements.** Specification of multi-character collating elements (that is, sequences of two or more characters to be collated as an entity).

2. **User-defined ordering of collating elements.** Each collating element shall be assigned a collation value defining its order in the character (or basic) collation sequence. This ordering is used by regular expressions and pattern matching and, unless collation weights are explicitly specified, also as the collation weight to be used in sorting.
3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or more (up to the limit {COLL_WEIGHTS_MAX}, as defined in **<limits.h>**) collating weights for use in sorting. The first weight is hereafter referred to as the primary weight.
4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
5. **Equivalence class definition.** Two or more collating elements have the same collation value (primary weight).
6. **Ordering by weights.** When two strings are compared to determine their relative order, the two strings are first broken up into a series of collating elements; the elements in each successive pair of elements are then compared according to the relative primary weights for the elements. If equal, and more than one weight has been assigned, then the pairs of collating elements are re-compared according to the relative subsequent weights, until either a pair of collating elements compare unequal or the weights are exhausted.

The following keywords shall be recognized in a collation sequence definition. They are described in detail in the following sections.

copy	Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.
collating-element	Define a collating-element symbol representing a multi-character collating element. This keyword is optional.
collating-symbol	Define a collating symbol for use in collation order statements. This keyword is optional.
order_start	Define collation rules. This statement shall be followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.
order_end	Specify the end of the collation-order statements.

7.3.2.1 The collating-element Keyword

In addition to the collating elements in the character set, the **collating-element** keyword can be used to define multi-character collating elements. The syntax is as follows:

```
"collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

The **<collating-symbol>** operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A **<collating-element>** defined via this keyword is only recognized with the **LC_COLLATE** category.

For example:

```
collating-element <ch> from "<c><h>"
collating-element <e-acute> from "<acute><e>"
collating-element <ll> from "ll"
```

4498 7.3.2.2 *The collating-symbol Keyword*

4499 This keyword shall be used to define symbols for use in collation sequence statements; that is,
 4500 between the **order_start** and the **order_end** keywords. The syntax is as follows:

4501 "collating-symbol %s\n", <collating-symbol>

4502 The <collating-symbol> shall be a symbolic name, enclosed between angle brackets ('<' and
 4503 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any
 4504 other symbolic name defined in this collation definition. A <collating-symbol> defined via this
 4505 keyword is only recognized within the *LC_COLLATE* category.

4506 For example:

4507 collating-symbol <UPPER_CASE>

4508 collating-symbol <HIGH>

4509 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative
 4510 position in the character order sequence. While such a symbolic name does not represent any
 4511 collating element, it can be used as a weight.

4512 7.3.2.3 *The order_start Keyword*

4513 The **order_start** keyword shall precede collation order entries and also define the number of
 4514 weights for this collation sequence definition and other collation rules. The syntax is as follows:

4515 "order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...

4516 The operands to the **order_start** keyword are optional. If present, the operands define rules to be
 4517 applied when strings are compared. The number of operands define how many weights each
 4518 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the
 4519 first operand defines rules to be applied when comparing strings using the first (primary)
 4520 weight; the second when comparing strings using the second weight, and so on. Operands shall
 4521 be separated by semicolons (';'). Each operand shall consist of one or more collation
 4522 directives, separated by commas (','),. If the number of operands exceeds the
 4523 {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following
 4524 directives shall be supported:

4525 **forward** Specifies that comparison operations for the weight level shall proceed from start
 4526 of string towards the end of string.

4527 **backward** Specifies that comparison operations for the weight level shall proceed from end of
 4528 string towards the beginning of string.

4529 **position** Specifies that comparison operations for the weight level shall consider the relative
 4530 position of elements in the strings not subject to **IGNORE**. The string containing
 4531 an element not subject to **IGNORE** after the fewest collating elements subject to
 4532 **IGNORE** from the start of the compare shall collate first. If both strings contain a
 4533 character not subject to **IGNORE** in the same relative position, the collating values
 4534 assigned to the elements shall determine the ordering. In case of equality,
 4535 subsequent characters not subject to **IGNORE** shall be considered in the same
 4536 manner.

4537 The directives **forward** and **backward** are mutually-exclusive.

4538 If no operands are specified, a single **forward** operand shall be assumed.

4539 For example:

4540 order_start forward;backward

4541 7.3.2.4 Collation Order

4542 The **order_start** keyword shall be followed by collating identifier entries. The syntax for the
4543 collating element entries is as follows:

4544 "%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...

4545 Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section
4546 7.3 (on page 124)), a <collating-element>, a <collating-symbol>, an ellipsis, or the special symbol
4547 **UNDEFINED**. The order in which collating elements are specified determines the character
4548 order sequence, such that each collating element shall compare less than the elements following
4549 it.

4550 A <collating-element> shall be used to specify multi-character collating elements, and indicates
4551 that the character sequence specified via the <collating-element> is to be collated as a unit and in
4552 the relative order specified by its place.

4553 A <collating-symbol> can be used to define a position in the relative order for use in weights. No
4554 weights shall be specified with a <collating-symbol>.

4555 The ellipsis symbol specifies that a sequence of characters shall collate according to their
4556 encoded character values. It shall be interpreted as indicating that all characters with a coded
4557 character set value higher than the value of the character in the preceding line, and lower than
4558 the coded character set value for the character in the following line, in the current coded
4559 character set, shall be placed in the character collation order between the previous and the
4560 following character in ascending order according to their coded character set values. An initial
4561 ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing
4562 ellipsis as if the following line specified the highest coded character set value in the current
4563 coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do
4564 not specify characters in the current coded character set. The use of the ellipsis symbol ties the
4565 definition to a specific coded character set and may preclude the definition from being portable
4566 between implementations.

4567 The symbol **UNDEFINED** shall be interpreted as including all coded character set values not
4568 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character
4569 collation order at the point indicated by the symbol, and in ascending order according to their
4570 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded
4571 character set contains characters not specified in this section, the utility shall issue a warning
4572 message and place such characters at the end of the character collation order.

4573 The optional operands for each collation-element shall be used to define the primary, secondary,
4574 or subsequent weights for the collating element. The first operand specifies the relative primary
4575 weight, the second the relative secondary weight, and so on. Two or more collation-elements can
4576 be assigned the same weight; they belong to the same “equivalence class” if they have the same
4577 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**
4578 are removed, unless the **position** collation directive is specified for the corresponding level with
4579 the **order_start** keyword. Then each successive pair of elements shall be compared according to
4580 the relative weights for the elements. If the two strings compare equal, the process shall be
4581 repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

4582 Weights shall be expressed as characters (in any of the forms specified in Section 7.3 (on page
4583 124)), <collating-symbol>s, <collating-element>s, an ellipsis, or the special symbol **IGNORE**. A
4584 single character, a <collating-symbol>, or a <collating-element> shall represent the relative position

in the character collating sequence of the character or symbol, rather than the character or characters themselves. Thus, rather than assigning absolute values to weights, a particular weight is expressed using the relative order value assigned to a collating element based on its order in the character collation sequence.

One-to-many mapping is indicated by specifying two or more concatenated characters or symbolic names. For example, if the `<eszt>` is given the string "`<s><s>`" as a weight, comparisons are performed as if all occurrences of the `<eszt>` are replaced by "`<s><s>`" (assuming that "`<s>`" has the collating weight "`<s>`"). If it is necessary to define `<eszt>` and "`<s><s>`" as an equivalence class, then a collating element must be defined for the string "`ss`".

All characters specified via an ellipsis shall by default be assigned unique weights, equal to the relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special symbol shall by default be assigned the same primary weight (that is, they belong to the same equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each character in the sequence shall have unique weights, equal to the relative order of their character in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

The special keyword **IGNORE** as a weight shall indicate that when strings are compared using the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that is, as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are subject to **IGNORE** in their primary weight form an equivalence class.

An empty operand shall be interpreted as the collating element itself.

For example, the order statement:

```
<a>      <a>;<a>
```

is equal to:

```
<a>
```

An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be interpreted as the value of each character defined by the ellipsis.

The collation order as defined in this section affects the interpretation of bracket expressions in regular expressions (see Section 9.3.5 (on page 172)).

For example:


```

4616      order_start    forward;backward
4617      UNDEFINED      IGNORE;IGNORE
4618      <LOW>
4619      <space>         <LOW>;<space>
4620      ...             <LOW>;...
4621      <a>              <a>;<a>
4622      <a-acute>        <a>;<a-acute>
4623      <a-grave>        <a>;<a-grave>
4624      <A>              <a>;<A>
4625      <A-acute>        <a>;<A-acute>
4626      <A-grave>        <a>;<A-grave>
4627      <ch>             <ch>;<ch>
4628      <Ch>             <ch>;<Ch>
4629      <s>              <s>;<s>
4630      <eszet>          "<s><s>";"<eszet><eszet>"
4631      order_end

```

4632 This example is interpreted as follows:

- 4633 1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or
4634 via the ellipsis) shall be ignored for collation purposes.
- 4635 2. All characters between <space> and 'a' shall have the same primary equivalence class
4636 and individual secondary weights based on their ordinal encoded values.
- 4637 3. All characters based on the uppercase or lowercase character 'a' belong to the same
4638 primary equivalence class.
- 4639 4. The multi-character collating element <ch> is represented by the collating symbol <ch>
4640 and belongs to the same primary equivalence class as the multi-character collating element
4641 <Ch>.

4642 7.3.2.5 The order_end Keyword

4643 The collating order entries shall be terminated with an **order_end** keyword.

4644 7.3.2.6 LC_COLLATE Category in the POSIX Locale

4645 The collation sequence definition of the POSIX locale follows; the code listing depicts the
4646 *localedef* input.

```

4647 LC_COLLATE
4648 # This is the POSIX locale definition for the LC_COLLATE category.
4649 # The order is the same as in the ASCII codeset.
4650 order_start forward
4651 <NUL>
4652 <SOH>
4653 <STX>
4654 <ETX>
4655 <EOT>
4656 <ENQ>
4657 <ACK>
4658 <alert>
4659 <backspace>
4660 <tab>
4661 <newline>

```

4662	<vertical-tab>
4663	<form-feed>
4664	<carriage-return>
4665	<SO>
4666	<SI>
4667	<DLE>
4668	<DC1>
4669	<DC2>
4670	<DC3>
4671	<DC4>
4672	<NAK>
4673	<SYN>
4674	<ETB>
4675	<CAN>
4676	
4677	<SUB>
4678	<ESC>
4679	<IS4>
4680	<IS3>
4681	<IS2>
4682	<IS1>
4683	<space>
4684	<exclamation-mark>
4685	<quotation-mark>
4686	<number-sign>
4687	<dollar-sign>
4688	<percent-sign>
4689	<ampersand>
4690	<apostrophe>
4691	<left-parenthesis>
4692	<right-parenthesis>
4693	<asterisk>
4694	<plus-sign>
4695	<comma>
4696	<hyphen>
4697	<period>
4698	<slash>
4699	<zero>
4700	<one>
4701	<two>
4702	<three>
4703	<four>
4704	<five>
4705	<six>
4706	<seven>
4707	<eight>
4708	<nine>
4709	<colon>
4710	<semicolon>
4711	<less-than-sign>
4712	<equals-sign>
4713	<greater-than-sign>

4714	<question-mark>
4715	<commercial-at>
4716	<A>
4717	
4718	<C>
4719	<D>
4720	<E>
4721	<F>
4722	<G>
4723	<H>
4724	<I>
4725	<J>
4726	<K>
4727	<L>
4728	<M>
4729	<N>
4730	<O>
4731	<P>
4732	<Q>
4733	<R>
4734	<S>
4735	<T>
4736	<U>
4737	<V>
4738	<W>
4739	<X>
4740	<Y>
4741	<Z>
4742	<left-square-bracket>
4743	<backslash>
4744	<right-square-bracket>
4745	<circumflex>
4746	<underscore>
4747	<grave-accent>
4748	<a>
4749	
4750	<c>
4751	<d>
4752	<e>
4753	<f>
4754	<g>
4755	<h>
4756	<i>
4757	<j>
4758	<k>
4759	<l>
4760	<m>
4761	<n>
4762	<o>
4763	<p>
4764	<q>
4765	<r>

```

4766      <s>
4767      <t>
4768      <u>
4769      <v>
4770      <w>
4771      <x>
4772      <y>
4773      <z>
4774      <left-curly-bracket>
4775      <vertical-line>
4776      <right-curly-bracket>
4777      <tilde>
4778      <DEL>
4779      order_end
4780      #
4781      END LC_COLLATE

```

4782 7.3.3 LC_MONETARY

4783 The *LC_MONETARY* category shall define the rules and symbols that are used to format
 4784 monetary numeric information.

4785 XSI This information is available through the *localeconv()* function and is used by the *strfmon()*
 4786 function.

4787 XSI Some of the information is also available in an alternative form via the *nl_langinfo()* function
 4788 (see CRNCYSTR in <langinfo.h>).

4789 The following items are defined in this category of the locale. The item names are the keywords
 4790 recognized by the *localedef* utility when defining a locale. They are also similar to the member
 4791 names of the **lconv** structure defined in <locale.h>; see <locale.h> for the exact symbols in the
 4792 header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the
 4793 empty string ("") for unspecified or size zero string items.

4794 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
 4795 Section 7.4 (on page 153). For some keywords, the strings can contain only integers. Keywords
 4796 that are not provided, string values set to the empty string (""), or integer keywords set to -1,
 4797 are used to indicate that the value is not available in the locale. The following keywords shall be
 4798 recognized:

4799 **copy** Specify the name of an existing locale which shall be used as the
 4800 definition of this category. If this keyword is specified, no other keyword
 4801 shall be specified.

4802 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4803 **int_curr_symbol** The international currency symbol. The operand shall be a four-character
 4804 string, with the first three characters containing the alphabetic
 4805 international currency symbol. The international currency symbol should
 4806 be chosen in accordance with those specified in the ISO 4217 standard.
 4807 The fourth character shall be the character used to separate the
 4808 international currency symbol from the monetary quantity.

4809 **currency_symbol** The string that shall be used as the local currency symbol.

4810 **mon_decimal_point** The operand is a string containing the symbol that shall be used as the
 4811 decimal delimiter (radix character) in monetary formatted quantities.

4812	mon_thousands_sep	The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities.
4813		
4814		
4815	mon_grouping	Define the size of each group of digits in formatted monetary quantities. The operand is a sequence of integers separated by semicolons. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not <code>-1</code> , then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is <code>-1</code> , then no further grouping shall be performed.
4816		
4817		
4818		
4819		
4820		
4821		
4822		
4823	positive_sign	A string that shall be used to indicate a non-negative-valued formatted monetary quantity.
4824		
4825	negative_sign	A string that shall be used to indicate a negative-valued formatted monetary quantity.
4826		
4827	int_frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using int_curr_symbol .
4828		
4829		
4830	frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using currency_symbol .
4831		
4832		
4833	p_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4834		
4835		
4836	p_sep_by_space	An integer set to 0 if no space separates the currency_symbol from the value for a monetary quantity with a non-negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4837		
4838		
4839		
4840	n_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4841		
4842		
4843	n_sep_by_space	An integer set to 0 if no space separates the currency_symbol from the value for a monetary quantity with a negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4844		
4845		
4846		
4847	p_sign_posn	An integer set to a value indicating the positioning of the positive_sign for a monetary quantity with a non-negative value. The following integer values shall be recognized for int_n_sign_posn , int_p_sign_posn , n_sign_posn , and p_sign_posn :
4848		
4849		
4850		
4851		0 Parentheses enclose the quantity and the currency_symbol .
4852		1 The sign string precedes the quantity and the currency_symbol .
4853		2 The sign string succeeds the quantity and the currency_symbol .
4854		3 The sign string precedes the currency_symbol .
4855		4 The sign string succeeds the currency_symbol .

4856	n_sign_posn	An integer set to a value indicating the positioning of the negative_sign
4857		for a negative formatted monetary quantity.
4858	int_p_cs_precedes	An integer set to 1 if the int_curr_symbol precedes the value for a
4859		monetary quantity with a non-negative value, and set to 0 if the symbol
4860		succeeds the value.
4861	int_n_cs_precedes	An integer set to 1 if the int_curr_symbol precedes the value for a
4862		monetary quantity with a negative value, and set to 0 if the symbol
4863		succeeds the value.
4864	int_p_sep_by_space	An integer set to 0 if no space separates the int_curr_symbol from the
4865		value for a monetary quantity with a non-negative value, set to 1 if a
4866		space separates the symbol from the value, and set to 2 if a space
4867		separates the symbol and the sign string, if adjacent.
4868	int_n_sep_by_space	An integer set to 0 if no space separates the int_curr_symbol from the
4869		value for a monetary quantity with a negative value, set to 1 if a space
4870		separates the symbol from the value, and set to 2 if a space separates the
4871		symbol and the sign string, if adjacent.
4872	int_p_sign_posn	An integer set to a value indicating the positioning of the positive_sign
4873		for a positive monetary quantity formatted with the international format.
4874	int_n_sign_posn	An integer set to a value indicating the positioning of the negative_sign
4875		for a negative monetary quantity formatted with the international format.

4876 7.3.3.1 LC_MONETARY Category in the POSIX Locale

4877 The monetary formatting definitions for the POSIX locale follow; the code listing depicting the
 4878 xsl *localedef* input, the table representing the same information with the addition of *localeconv()* and
 4879 *nl_langinfo()* formats. All values are unspecified in the POSIX locale.

```

4880 LC_MONETARY
4881 # This is the POSIX locale definition for
4882 # the LC_MONETARY category.
4883 #
4884 int_curr_symbol      " "
4885 currency_symbol      " "
4886 mon_decimal_point    " "
4887 mon_thousands_sep   " "
4888 mon_grouping         -1
4889 positive_sign        " "
4890 negative_sign        " "
4891 int_frac_digits      -1
4892 frac_digits          -1
4893 p_cs_precedes        -1
4894 p_sep_by_space       -1
4895 n_cs_precedes        -1
4896 n_sep_by_space       -1
4897 p_sign_posn          -1
4898 n_sign_posn          -1
4899 int_p_cs_precedes    -1
4900 int_p_sep_by_space   -1
4901 int_n_cs_precedes    -1
4902 int_n_sep_by_space   -1

```

```

4903 int_p_sign_posn      -1
4904 int_n_sign_posn      -1
4905 #
4906 END LC_MONETARY

```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
int_curr_symbol	—	N/A	" "	" "
currency_symbol	CRNCYSTR	N/A	" "	" "
mon_decimal_point	—	N/A	" "	" "
mon_thousands_sep	—	N/A	" "	" "
mon_grouping	—	N/A	" "	–1
positive_sign	—	N/A	" "	" "
negative_sign	—	N/A	" "	" "
int_frac_digits	—	N/A	{CHAR_MAX}	–1
frac_digits	—	N/A	{CHAR_MAX}	–1
p_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	–1
p_sep_by_space	—	N/A	{CHAR_MAX}	–1
n_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	–1
n_sep_by_space	—	N/A	{CHAR_MAX}	–1
p_sign_posn	—	N/A	{CHAR_MAX}	–1
n_sign_posn	—	N/A	{CHAR_MAX}	–1
int_p_cs_precedes	—	N/A	{CHAR_MAX}	–1
int_p_sep_by_space	—	N/A	{CHAR_MAX}	–1
int_n_cs_precedes	—	N/A	{CHAR_MAX}	–1
int_n_sep_by_space	—	N/A	{CHAR_MAX}	–1
int_p_sign_posn	—	N/A	{CHAR_MAX}	–1
int_n_sign_posn	—	N/A	{CHAR_MAX}	–1

4930 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.
 4931 The entry N/A indicates that the value is not available in the POSIX locale.

4932 7.3.4 LC_NUMERIC

4933 The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-
 4934 monetary numeric information. This information is available through the *localeconv()* function.

4935 XSI Some of the information is also available in an alternative form via the *nl_langinfo()* function.

4936 The following items are defined in this category of the locale. The item names are the keywords
 4937 recognized by the *localedef* utility when defining a locale. They are also similar to the member
 4938 names of the **lconv** structure defined in **<locale.h>**; see **<locale.h>** for the exact symbols in the
 4939 header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the
 4940 empty string (" ") for unspecified or size zero string items.

4941 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
 4942 Section 7.4 (on page 153). For some keywords, the strings can only contain integers. Keywords
 4943 that are not provided, string values set to the empty string (" "), or integer keywords set to –1,
 4944 shall be used to indicate that the value is not available in the locale. The following keywords
 4945 shall be recognized:

4946 **copy** Specify the name of an existing locale which shall be used as the definition of
 4947 this category. If this keyword is specified, no other keyword shall be specified.

4948 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4949 **decimal_point** The operand is a string containing the symbol that shall be used as the
 4950 decimal delimiter (radix character) in numeric, non-monetary formatted
 4951 quantities. This keyword cannot be omitted and cannot be set to the empty
 4952 string. In contexts where standards limit the **decimal_point** to a single byte,
 4953 the result of specifying a multi-byte operand shall be unspecified.

4954 **thousands_sep** The operand is a string containing the symbol that shall be used as a separator
 4955 for groups of digits to the left of the decimal delimiter in numeric, non-
 4956 monetary formatted monetary quantities. In contexts where standards limit
 4957 the **thousands_sep** to a single byte, the result of specifying a multi-byte
 4958 operand shall be unspecified.

4959 **grouping** Define the size of each group of digits in formatted non-monetary quantities.
 4960 The operand is a sequence of integers separated by semicolons. Each integer
 4961 specifies the number of digits in each group, with the initial integer defining
 4962 the size of the group immediately preceding the decimal delimiter, and the
 4963 following integers defining the preceding groups. If the last integer is not -1,
 4964 then the size of the previous group (if any) shall be repeatedly used for the
 4965 remainder of the digits. If the last integer is -1, then no further grouping shall
 4966 be performed.

4967 7.3.4.1 LC_NUMERIC Category in the POSIX Locale

4968 The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing
 4969 depicting the *localedef* input, the table representing the same information with the addition of
 4970 XSI *localeconv()* values, and *nl_langinfo()* constants.

```

4971 LC_NUMERIC
4972 # This is the POSIX locale definition for
4973 # the LC_NUMERIC category.
4974 #
4975 decimal_point      "<period>"
4976 thousands_sep      ""
4977 grouping           -1
4978 #
4979 END LC_NUMERIC

```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
decimal_point	RADIXCHAR	."	."	.
thousands_sep	THOUSEP	N/A	""	""
grouping	—	N/A	""	-1

4985 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conforming extension.
 4986 The entry N/A indicates that the value is not available in the POSIX locale.

4987 7.3.5 LC_TIME

4988 The *LC_TIME* category shall define the interpretation of the conversion specifications supported
 4989 XSI by the *date* utility and shall affect the behavior of the *strftime()*, *wcsftime()*, *strptime()*, and
 4990 *nl_langinfo()* functions. Since the interfaces for C-language access and locale definition differ
 4991 significantly, they are described separately.

4992 7.3.5.1 LC_TIME Locale Definition

4993 In a locale definition, the following mandatory keywords shall be recognized:

4994	copy	Specify the name of an existing locale which shall be used as the definition of
4995		this category. If this keyword is specified, no other keyword shall be specified.
4996	abday	Define the abbreviated weekday names, corresponding to the %a conversion
4997		specification (conversion specification in the <i>strftime()</i> , <i>wcsftime()</i> , and
4998		<i>strptime()</i> functions). The operand shall consist of seven semicolon-separated
4999		strings, each surrounded by double-quotes. The first string shall be the
5000		abbreviated name of the day corresponding to Sunday, the second the
5001		abbreviated name of the day corresponding to Monday, and so on.
5002	day	Define the full weekday names, corresponding to the %A conversion
5003		specification. The operand shall consist of seven semicolon-separated strings,
5004		each surrounded by double-quotes. The first string is the full name of the day
5005		corresponding to Sunday, the second the full name of the day corresponding
5006		to Monday, and so on.
5007	abmon	Define the abbreviated month names, corresponding to the %b conversion
5008		specification. The operand shall consist of twelve semicolon-separated strings,
5009		each surrounded by double-quotes. The first string shall be the abbreviated
5010		name of the first month of the year (January), the second the abbreviated
5011		name of the second month, and so on.
5012	mon	Define the full month names, corresponding to the %B conversion
5013		specification. The operand shall consist of twelve semicolon-separated strings,
5014		each surrounded by double-quotes. The first string shall be the full name of
5015		the first month of the year (January), the second the full name of the second
5016		month, and so on.
5017	d_t_fmt	Define the appropriate date and time representation, corresponding to the %c
5018		conversion specification. The operand shall consist of a string containing any
5019		combination of characters and conversion specifications. In addition, the
5020		string can contain escape sequences defined in the table in Table 5-1 (on page
5021		112) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').
5022	d_fmt	Define the appropriate date representation, corresponding to the %x
5023		conversion specification. The operand shall consist of a string containing any
5024		combination of characters and conversion specifications. In addition, the
5025		string can contain escape sequences defined in Table 5-1 (on page 112).
5026	t_fmt	Define the appropriate time representation, corresponding to the %X
5027		conversion specification. The operand shall consist of a string containing any
5028		combination of characters and conversion specifications. In addition, the
5029		string can contain escape sequences defined in Table 5-1 (on page 112).
5030	am_pm	Define the appropriate representation of the <i>ante-meridiem</i> and <i>post-meridiem</i>
5031		strings, corresponding to the %p conversion specification. The operand shall
5032		consist of two strings, separated by a semicolon, each surrounded by double-

5033		quotes. The first string shall represent the <i>ante-meridiem</i> designation, the last string the <i>post-meridiem</i> designation.
5034		
5035	t_fmt_ampm	Define the appropriate time representation in the 12-hour clock format with am_pm , corresponding to the %r conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If the string is empty, the 12-hour format is not supported in the locale.
5036		
5037		
5038		
5039		
5040	era	Define how years are counted and displayed for each era in a locale. The operand shall consist of semicolon-separated strings. Each string shall be an era description segment with the format:
5041		
5042		
5043		<i>direction:offset:start_date:end_date:era_name:era_format</i>
5044		according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.
5045		
5046	Note:	The start of an era might not be the earliest point in the era—it may be the latest. For example, the Christian era BC starts on the day before January 1, AD 1, and increases with earlier time.
5047		
5048		
5049	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character shall indicate that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
5050		
5051		
5052		
5053		
5054	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era, corresponding to the %Ey conversion specification.
5055		
5056	<i>start_date</i>	A date in the form yyyy/mm/dd, where yyyy, mm, and dd are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.
5057		
5058		
5059		
5060	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.
5061		
5062		
5063		
5064	<i>era_name</i>	A string representing the name of the era, corresponding to the %EC conversion specification.
5065		
5066	<i>era_format</i>	A string for formatting the year in the era, corresponding to the %EY conversion specification.
5067		
5068	era_d_fmt	Define the format of the date in alternative era notation, corresponding to the %Ex conversion specification.
5069		
5070	era_t_fmt	Define the locale's appropriate alternative time format, corresponding to the %EX conversion specification.
5071		
5072	era_d_t_fmt	Define the locale's appropriate alternative date and time format, corresponding to the %Ec conversion specification.
5073		
5074	alt_digits	Define alternative symbols for digits, corresponding to the %O modified conversion specification. The operand shall consist of semicolon-separated strings, each surrounded by double-quotes. The first string shall be the alternative symbol corresponding with zero, the second string the symbol
5075		
5076		
5077		

5078 corresponding with one, and so on. Up to 100 alternative symbol strings can
 5079 be specified. The %O modifier shall indicate that the string corresponding to
 5080 the value specified via the conversion specification shall be used instead of the
 5081 value.

5082 7.3.5.2 LC_TIME C-Language Access

5083 XSI This section describes extensions to access information in the *LC_TIME* category using the
 5084 *nl_langinfo()* function. This functionality is dependent on support of the XSI extension (and the
 5085 rest of this section is not further shaded for this option).

5086 The following constants used to identify items of *langinfo* data can be used as arguments to the
 5087 *nl_langinfo()* function to access information in the *LC_TIME* category. These constants are
 5088 defined in the **<langinfo.h>** header.

5089 ABDAY_x The abbreviated weekday names (for example, Sun), where *x* is a number from
 5090 1 to 7.

5091 DAY_x The full weekday names (for example, Sunday), where *x* is a number from 1 to
 5092 7.

5093 ABMON_x The abbreviated month names (for example, Jan), where *x* is a number from 1
 5094 to 12.

5095 MON_x The full month names (for example, January), where *x* is a number from 1 to
 5096 12.

5097 D_T_FMT The appropriate date and time representation.

5098 D_FMT The appropriate date representation.

5099 T_FMT The appropriate time representation.

5100 AM_STR The appropriate ante-meridiem affix.

5101 PM_STR The appropriate post-meridiem affix.

5102 T_FMT_AMPM The appropriate time representation in the 12-hour clock format with
 5103 AM_STR and PM_STR.

5104 ERA The era description segments, which describe how years are counted and
 5105 displayed for each era in a locale. Each era description segment shall have the
 5106 format:

5107 *direction:offset:start_date:end_date:era_name:era_format*

5108 according to the definitions below. There can be as many era description
 5109 segments as are necessary to describe the different eras. Era description
 5110 segments are separated by semicolons.

5111 *direction* Either a '+' or a '-' character. The '+' character shall indicate
 5112 that years closer to the *start_date* have lower numbers than those
 5113 closer to the *end_date*. The '-' character shall indicate that
 5114 years closer to the *start_date* have higher numbers than those
 5115 closer to the *end_date*.

5116 *offset* The number of the year closest to the *start_date* in the era.

5117 *start_date* A date in the form *yyyy/mm/dd*, where *yyyy*, *mm*, and *dd* are the
 5118 year, month, and day numbers respectively of the start of the
 5119 era. Years prior to AD 1 shall be represented as negative

5120		numbers.
5121	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> ,
5122		or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The
5123		value "+*" shall indicate that the ending date is the end of time.
5124		
5125	<i>era_name</i>	The era, corresponding to the %EC conversion specification.
5126	<i>era_format</i>	The format of the year in the era, corresponding to the %EY conversion specification.
5127		
5128	ERA_D_FMT	The era date format.
5129	ERA_T_FMT	The locale's appropriate alternative time format, corresponding to the %EX conversion specification.
5130		
5131	ERA_D_T_FMT	The locale's appropriate alternative date and time format, corresponding to the %Ec conversion specification.
5132		
5133	ALT_DIGITS	The alternative symbols for digits, corresponding to the %O conversion specification modifier. The value consists of semicolon-separated symbols. The first is the alternative symbol corresponding to zero, the second is the symbol corresponding to one, and so on. Up to 100 alternative symbols may be specified.
5134		
5135		
5136		
5137		

5138 7.3.5.3 LC_TIME Category in the POSIX Locale

5139 The *LC_TIME* category definition of the POSIX locale follows; the code listing depicts the
 5140 *localedef* input; the table represents the same information with the addition of *localedef* keywords,
 5141 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*
 5142 XSI functions, and *nl_langinfo()* constants.

```

5143 LC_TIME
5144 # This is the POSIX locale definition for
5145 # the LC_TIME category.
5146 #
5147 # Abbreviated weekday names (%a)
5148 abday      "<S><u><n>"; "<M><o><n>"; "<T><u><e>"; "<W><e><d>"; \
5149            "<T><h><u>"; "<F><r><i>"; "<S><a><t>"
5150 #
5151 # Full weekday names (%A)
5152 day        "<S><u><n><d><a><y>"; "<M><o><n><d><a><y>"; \
5153            "<T><u><e><s><d><a><y>"; "<W><e><d><n><e><s><d><a><y>"; \
5154            "<T><h><u><r><s><d><a><y>"; "<F><r><i><d><a><y>"; \
5155            "<S><a><t><u><r><d><a><y>"
5156 #
5157 # Abbreviated month names (%b)
5158 abmon      "<J><a><n>"; "<F><e><b>"; "<M><a><r>"; \
5159            "<A><p><r>"; "<M><a><y>"; "<J><u><n>"; \
5160            "<J><u><l>"; "<A><u><g>"; "<S><e><p>"; \
5161            "<O><c><t>"; "<N><o><v>"; "<D><e><c>"
5162 #
5163 # Full month names (%B)
5164 mon        "<J><a><n><u><a><r><y>"; "<F><e><b><r><u><a><r><y>"; \
5165            "<M><a><r><c><h>"; "<A><p><r><i><l>"; \
5166            "<M><a><y>"; "<J><u><n><e>"; \

```

```

5167         "<J><u><l><y>"; "<A><u><g><u><s><t>"; \
5168         "<S><e><p><t><e><m><b><e><r>"; "<O><c><t><o><b><e><r>"; \
5169         "<N><o><v><e><m><b><e><r>"; "<D><e><c><e><m><b><e><r>"
5170     #
5171     # Equivalent of AM/PM (%p)          "AM"; "PM"
5172     am_pm          "<A><M>"; "<P><M>"
5173     #
5174     # Appropriate date and time representation (%c)
5175     #      "%a %b %e %H:%M:%S %Y"
5176     d_t_fmt        "<percent-sign><a><space><percent-sign><b>\
5177     <space><percent-sign><e><space><percent-sign><H>\
5178     <colon><percent-sign><M><colon><percent-sign><S>\
5179     <space><percent-sign><Y>"
5180     #
5181     # Appropriate date representation (%x)  "%m/%d/%y"
5182     d_fmt          "<percent-sign><m><slash><percent-sign><d>\
5183     <slash><percent-sign><y>"
5184     #
5185     # Appropriate time representation (%X)  "%H:%M:%S"
5186     t_fmt          "<percent-sign><H><colon><percent-sign><M>\
5187     <colon><percent-sign><S>"
5188     #
5189     # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5190     t_fmt_ampm     "<percent-sign><I><colon><percent-sign><M><colon>\
5191     <percent-sign><S><space><percent_sign><p>"
5192     #
5193     END LC_TIME

```

5194

5195

5196

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
d_fmt	D_FMT	%x	"%m/%d/%y"
t_fmt	T_FMT	%X	"%H:%M:%S"
am_pm	AM_STR	%p	"AM"
am_pm	PM_STR	%p	"PM"
t_fmt_ampm	T_FMT_AMP	%r	"%I:%M:%S %p"
day	DAY_1	%A	"Sunday"
day	DAY_2	%A	"Monday"
day	DAY_3	%A	"Tuesday"
day	DAY_4	%A	"Wednesday"
day	DAY_5	%A	"Thursday"
day	DAY_6	%A	"Friday"
day	DAY_7	%A	"Saturday"
abday	ABDAY_1	%a	"Sun"
abday	ABDAY_2	%a	"Mon"
abday	ABDAY_3	%a	"Tue"
abday	ABDAY_4	%a	"Wed"
abday	ABDAY_5	%a	"Thu"

5214

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
abday	ABDAY_6	%a	"Fri "
abday	ABDAY_7	%a	"Sat "
mon	MON_1	%B	"January"
mon	MON_2	%B	"February"
mon	MON_3	%B	"March"
mon	MON_4	%B	"April "
mon	MON_5	%B	"May"
mon	MON_6	%B	"June"
mon	MON_7	%B	"July"
mon	MON_8	%B	"August "
mon	MON_9	%B	"September"
mon	MON_10	%B	"October"
mon	MON_11	%B	"November"
mon	MON_12	%B	"December"
abmon	ABMON_1	%b	"Jan"
abmon	ABMON_2	%b	"Feb"
abmon	ABMON_3	%b	"Mar"
abmon	ABMON_4	%b	"Apr"
abmon	ABMON_5	%b	"May"
abmon	ABMON_6	%b	"Jun"
abmon	ABMON_7	%b	"Jul "
abmon	ABMON_8	%b	"Aug"
abmon	ABMON_9	%b	"Sep"
abmon	ABMON_10	%b	"Oct "
abmon	ABMON_11	%b	"Nov"
abmon	ABMON_12	%b	"Dec"
era	ERA	%EC, %Ey, %EY	N/A
era_d_fmt	ERA_D_FMT	%Ex	N/A
era_t_fmt	ERA_T_FMT	%EX	N/A
era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
alt_digits	ALT_DIGITS	%O	N/A

In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

The entry N/A indicates the value is not available in the POSIX locale.

7.3.6 LC_MESSAGES

The *LC_MESSAGES* category shall define the format and values used by various utilities for affirmative and negative responses. This information is available through the *nl_langinfo()* function.

The message catalog used by the standard utilities and selected by the *catopen()* function shall be determined by the setting of *NLSPATH*; see Chapter 8 (on page 161). The *LC_MESSAGES* category can be specified as part of an *NLSPATH* substitution field.

The following keywords shall be recognized as part of the locale definition file.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* keyword, unavailable through *nl_langinfo()*.

5262 **yesexpr** The operand consists of an extended regular expression (see Section 9.4 (on page
5263 175)) that describes the acceptable affirmative response to a question expecting an
5264 affirmative or negative response.

5265 **noexpr** The operand consists of an extended regular expression that describes the
5266 acceptable negative response to a question expecting an affirmative or negative
5267 response.

5268 7.3.6.1 *LC_MESSAGES Category in the POSIX Locale*

5269 The format and values for affirmative and negative responses of the POSIX locale follow; the
5270 XSI code listing depicting the *localedef* input, the table representing the same information with the
5271 addition of *nl_langinfo()* constants.

```
5272 LC_MESSAGES
5273 # This is the POSIX locale definition for
5274 # the LC_MESSAGES category.
5275 #
5276 yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5277 #
5278 noexpr  "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5279 #
5280 END LC_MESSAGES
```

5281	localedef Keyword	langinfo Constant	POSIX Locale Value
5282	yesexpr	YESEXPR	"^[yY]"
5283	noexpr	NOEXPR	"^[nN]"

5284 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

5285 7.4 Locale Definition Grammar

5286 The grammar and lexical conventions in this section shall together describe the syntax for the
5287 locale definition source. The general conventions for this style of grammar are described in the
5288 Shell and Utilities volume of IEEE Std 1003.1-2001, Section 1.10, Grammar Conventions. The
5289 grammar shall take precedence over the text in this chapter.

5290 7.4.1 Locale Lexical Conventions

5291 The lexical conventions for the locale definition grammar are described in this section.

5292 The following tokens shall be processed (in addition to those string constants shown in the
5293 grammar):

5294	LOC_NAME	A string of characters representing the name of a locale.
5295	CHAR	Any single character.
5296	NUMBER	A decimal number, represented by one or more decimal digits.
5297	COLLSYMBOL	A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a COLLELEMENT symbol.
5298		
5299		
5300	COLLELEMENT	A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a COLLSYMBOL symbol.
5301		

5302	CHARCLASS	A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes.
5303		
5304		
5305		
5306	CHARSYMBOL	A symbolic name, enclosed between angle brackets, from the current charmap (if any).
5307		
5308	OCTAL_CHAR	One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a backslash) followed by two or more octal digits.
5309		
5310		
5311		
5312	HEX_CHAR	One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant <i>x</i> and two or more hexadecimal digits.
5313		
5314		
5315		
5316	DECIMAL_CHAR	One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character 'd' and two or more decimal digits.
5317		
5318		
5319		
5320	ELLIPSIS	The string ". . .".
5321	EXTENDED_REG_EXP	An extended regular expression as defined in the grammar in Section 9.5 (on page 179).
5322		
5323	EOL	The line termination character <newline>.

5324 7.4.2 Locale Grammar

5325 This section presents the grammar for the locale definition.

5326	%token	LOC_NAME
5327	%token	CHAR
5328	%token	NUMBER
5329	%token	COLLSYMBOL COLLELEMENT
5330	%token	CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5331	%token	ELLIPSIS
5332	%token	EXTENDED_REG_EXP
5333	%token	EOL
5334	%start	locale_definition
5335	%%	
5336	locale_definition	: global_statements locale_categories
5337		locale_categories
5338		;
5339	global_statements	: global_statements symbol_redefine
5340		symbol_redefine
5341		;
5342	symbol_redefine	: 'escape_char' CHAR EOL
5343		'comment_char' CHAR EOL
5344		;


```

5345     locale_categories      : locale_categories locale_category
5346                             | locale_category
5347                             ;
5348     locale_category         : lc_ctype | lc_collate | lc_messages
5349                             | lc_monetary | lc_numeric | lc_time
5350                             ;
5351     /* The following grammar rules are common to all categories */
5352     char_list                : char_list char_symbol
5353                             | char_symbol
5354                             ;
5355     char_symbol              : CHAR | CHARSYMBOL
5356                             | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5357                             ;
5358     elem_list                : elem_list char_symbol
5359                             | elem_list COLLSYMBOL
5360                             | elem_list COLLELEMENT
5361                             | char_symbol
5362                             | COLLSYMBOL
5363                             | COLLELEMENT
5364                             ;
5365     symb_list                : symb_list COLLSYMBOL
5366                             | COLLSYMBOL
5367                             ;
5368     locale_name              : LOC_NAME
5369                             | '"" LOC_NAME ""'
5370                             ;
5371     /* The following is the LC_CTYPE category grammar */
5372     lc_ctype                  : ctype_hdr ctype_keywords          ctype_tlr
5373                             | ctype_hdr 'copy' locale_name EOL ctype_tlr
5374                             ;
5375     ctype_hdr                 : 'LC_CTYPE' EOL
5376                             ;
5377     ctype_keywords            : ctype_keywords ctype_keyword
5378                             | ctype_keyword
5379                             ;
5380     ctype_keyword             : charclass_keyword charclass_list EOL
5381                             | charconv_keyword charconv_list EOL
5382                             | 'charclass' charclass_namelist EOL
5383                             ;
5384     charclass_namelist        : charclass_namelist ';' CHARCLASS
5385                             | CHARCLASS
5386                             ;
5387     charclass_keyword         : 'upper' | 'lower' | 'alpha' | 'digit'
5388                             | 'punct' | 'xdigit' | 'space' | 'print'
5389                             | 'graph' | 'blank' | 'cntrl' | 'alnum'

```

```

5390          | CHARCLASS
5391          ;
5392      charclass_list      : charclass_list ';' char_symbol
5393          | charclass_list ';' ELLIPSIS ';' char_symbol
5394          | char_symbol
5395          ;
5396      charconv_keyword    : 'toupper'
5397          | 'tolower'
5398          ;
5399      charconv_list       : charconv_list ';' charconv_entry
5400          | charconv_entry
5401          ;
5402      charconv_entry      : '(' char_symbol ',' char_symbol ')'
5403          ;
5404      ctype_tlr           : 'END' 'LC_CTYPE' EOL
5405          ;
5406      /* The following is the LC_COLLATE category grammar */
5407      lc_collate           : collate_hdr collate_keywords      collate_tlr
5408          | collate_hdr 'copy' locale_name EOL collate_tlr
5409          ;
5410      collate_hdr         : 'LC_COLLATE' EOL
5411          ;
5412      collate_keywords    :          order_statements
5413          | opt_statements order_statements
5414          ;
5415      opt_statements      : opt_statements collating_symbols
5416          | opt_statements collating_elements
5417          | collating_symbols
5418          | collating_elements
5419          ;
5420      collating_symbols   : 'collating-symbol' COLLSYMBOL EOL
5421          ;
5422      collating_elements  : 'collating-element' COLLELEMENT
5423          | 'from' '"' elem_list '"' EOL
5424          ;
5425      order_statements    : order_start collation_order order_end
5426          ;
5427      order_start         : 'order_start' EOL
5428          | 'order_start' order_opts EOL
5429          ;
5430      order_opts          : order_opts ';' order_opt
5431          | order_opt
5432          ;

```

```

5433     order_opt           : order_opt ',' opt_word
5434                           | opt_word
5435                           ;
5436     opt_word             : 'forward' | 'backward' | 'position'
5437                           ;
5438     collation_order      : collation_order collation_entry
5439                           | collation_entry
5440                           ;
5441     collation_entry      : COLLSYMBOL EOL
5442                           | collation_element weight_list EOL
5443                           | collation_element EOL
5444                           ;
5445     collation_element    : char_symbol
5446                           | COLLELEMENT
5447                           | ELLIPSIS
5448                           | 'UNDEFINED'
5449                           ;
5450     weight_list          : weight_list ';' weight_symbol
5451                           | weight_list ';'
5452                           | weight_symbol
5453                           ;
5454     weight_symbol        : /* empty */
5455                           | char_symbol
5456                           | COLLSYMBOL
5457                           | ''' elem_list '''
5458                           | ''' symb_list '''
5459                           | ELLIPSIS
5460                           | 'IGNORE'
5461                           ;
5462     order_end            : 'order_end' EOL
5463                           ;
5464     collate_tlr          : 'END' 'LC_COLLATE' EOL
5465                           ;
5466     /* The following is the LC_MESSAGES category grammar */
5467     lc_messages          : messages_hdr messages_keywords messages_tlr
5468                           | messages_hdr 'copy' locale_name EOL messages_tlr
5469                           ;
5470     messages_hdr         : 'LC_MESSAGES' EOL
5471                           ;
5472     messages_keywords    : messages_keywords messages_keyword
5473                           | messages_keyword
5474                           ;
5475     messages_keyword     : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5476                           | 'noexpr' ''' EXTENDED_REG_EXP ''' EOL
5477                           ;

```

```

5478     messages_tlr           : 'END' 'LC_MESSAGES' EOL
5479                             ;

5480     /* The following is the LC_MONETARY category grammar */

5481     lc_monetary              : monetary_hdr monetary_keywords      monetary_tlr
5482                             | monetary_hdr 'copy' locale_name EOL  monetary_tlr
5483                             ;

5484     monetary_hdr             : 'LC_MONETARY' EOL
5485                             ;

5486     monetary_keywords        : monetary_keywords monetary_keyword
5487                             | monetary_keyword
5488                             ;

5489     monetary_keyword          : mon_keyword_string mon_string EOL
5490                             | mon_keyword_char NUMBER EOL
5491                             | mon_keyword_char '-1' EOL
5492                             | mon_keyword_grouping mon_group_list EOL
5493                             ;

5494     mon_keyword_string        : 'int_curr_symbol' | 'currency_symbol'
5495                             | 'mon_decimal_point' | 'mon_thousands_sep'
5496                             | 'positive_sign' | 'negative_sign'
5497                             ;

5498     mon_string                : '"' char_list '"'
5499                             | '"""'
5500                             ;

5501     mon_keyword_char          : 'int_frac_digits' | 'frac_digits'
5502                             | 'p_cs_precedes' | 'p_sep_by_space'
5503                             | 'n_cs_precedes' | 'n_sep_by_space'
5504                             | 'p_sign_posn' | 'n_sign_posn'
5505                             | 'int_p_cs_precedes' | 'int_p_sep_by_space'
5506                             | 'int_n_cs_precedes' | 'int_n_sep_by_space'
5507                             | 'int_p_sign_posn' | 'int_n_sign_posn'
5508                             ;

5509     mon_keyword_grouping      : 'mon_grouping'
5510                             ;

5511     mon_group_list            : NUMBER
5512                             | mon_group_list ';' NUMBER
5513                             ;

5514     monetary_tlr              : 'END' 'LC_MONETARY' EOL
5515                             ;

5516     /* The following is the LC_NUMERIC category grammar */

5517     lc_numeric                : numeric_hdr numeric_keywords      numeric_tlr
5518                             | numeric_hdr 'copy' locale_name EOL  numeric_tlr
5519                             ;

5520     numeric_hdr               : 'LC_NUMERIC' EOL
5521                             ;

```

```

5522     numeric_keywords      : numeric_keywords numeric_keyword
5523                             | numeric_keyword
5524                             ;
5525     numeric_keyword        : num_keyword_string num_string EOL
5526                             | num_keyword_grouping num_group_list EOL
5527                             ;
5528     num_keyword_string     : 'decimal_point'
5529                             | 'thousands_sep'
5530                             ;
5531     num_string             : ''' char_list '''
5532                             | '""'
5533                             ;
5534     num_keyword_grouping   : 'grouping'
5535                             ;
5536     num_group_list         : NUMBER
5537                             | num_group_list ';' NUMBER
5538                             ;
5539     numeric_tlr            : 'END' 'LC_NUMERIC' EOL
5540                             ;
5541     /* The following is the LC_TIME category grammar */
5542     lc_time                 : time_hdr time_keywords          time_tlr
5543                             | time_hdr 'copy' locale_name EOL time_tlr
5544                             ;
5545     time_hdr               : 'LC_TIME' EOL
5546                             ;
5547     time_keywords          : time_keywords time_keyword
5548                             | time_keyword
5549                             ;
5550     time_keyword           : time_keyword_name time_list EOL
5551                             | time_keyword_fmt time_string EOL
5552                             | time_keyword_opt time_list EOL
5553                             ;
5554     time_keyword_name      : 'abday' | 'day' | 'abmon' | 'mon'
5555                             ;
5556     time_keyword_fmt       : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5557                             | 'am_pm' | 't_fmt_ampm'
5558                             ;
5559     time_keyword_opt       : 'era' | 'era_d_fmt' | 'era_t_fmt'
5560                             | 'era_d_t_fmt' | 'alt_digits'
5561                             ;
5562     time_list              : time_list ';' time_string
5563                             | time_string
5564                             ;

```

```
5565         time_string          : '""' char_list '""'
5566                               ;
5567         time_tlr               : 'END' 'LC_TIME' EOL
5568                               ;
```

Environment Variables

5569

8.1 Environment Variable Definition

Environment variables defined in this chapter affect the operation of multiple utilities, functions, and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in the Shell and Utilities volume of IEEE Std 1003.1-2001 for information on environment variable usage.

The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable *environ*, which is defined as:

```
extern char **environ;
```

These strings have the form *name=value*; *names* shall not contain the character '='. For values to be portable across systems conforming to IEEE Std 1003.1-2001, the value shall be composed of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in a process' environment has the same *name*, the consequences are undefined.

Environment variable names used by the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 consist solely of uppercase letters, digits, and the '_' (underscore) from the characters defined in Table 6-1 (on page 115) and do not begin with a digit. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for applications. Applications can define any environment variables with names from this name space without modifying the behavior of the standard utilities.

Note: Other applications may have difficulty dealing with environment variable names that start with a digit. For this reason, use of such names is not recommended anywhere.

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG_MAX} bytes.

Other *name=value* pairs may be placed in the environment by, for example, calling any of the *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of IEEE Std 1003.1-2001.

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

5605	ARFLAGS	IFS	MAILPATH	PS1
5606	CC	LANG	MAILRC	PS2
5607	CDPATH	LC_ALL	MAKEFLAGS	PS3
5608	CFLAGS	LC_COLLATE	MAKESHELL	PS4
5609	CHARSET	LC_CTYPE	MANPATH	PWD
5610	COLUMNS	LC_MESSAGES	MBOX	RANDOM
5611	DATMSK	LC_MONETARY	MORE	SECONDS
5612	DEAD	LC_NUMERIC	MSGVERB	SHELL
5613	EDITOR	LC_TIME	NLSPATH	TERM
5614	ENV	LDFLAGS	NPROC	TERMCAP
5615	EXINIT	LEX	OLDPWD	TERMINFO
5616	FC	LFLAGS	OPTARG	TMPDIR
5617	FCEDIT	LINENO	OPTERR	TZ
5618	FFLAGS	LINES	OPTIND	USER
5619	GET	LISTER	PAGER	VISUAL
5620	GFLAGS	LOGNAME	PATH	YACC
5621	HISTFILE	LPDEST	PPID	YFLAGS
5622	HISTORY	MAIL	PRINTER	
5623	HISTSIZE	MAILCHECK	PROCLANG	
5624	HOME	MAILER	PROJECTDIR	

If the variables in the following two sections are present in the environment during the execution of an application or utility, they shall be given the meaning described below. Some are placed into the environment by the implementation at the time the user logs in; all can be added or changed by the user or any ancestor of the current process. The implementation adds or changes environment variables named in IEEE Std 1003.1-2001 only as specified in IEEE Std 1003.1-2001. If they are defined in the application's environment, the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 and the functions in the System Interfaces volume of IEEE Std 1003.1-2001 assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See *getenv()* and the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.12, Shell Execution Environment for methods of accessing these variables.

8.2 Internationalization Variables

This section describes environment variables that are relevant to the operation of internationalized interfaces described in IEEE Std 1003.1-2001.

Users may use the following environment variables to announce specific localization requirements to applications. Applications can retrieve this information using the *setlocale()* function to initialize the correct behavior of the internationalized interfaces. The descriptions of the internationalization environment variables describe the resulting behavior only when the application locale is initialized in this way. The use of the internationalization variables by utilities described in the Shell and Utilities volume of IEEE Std 1003.1-2001 is described in the ENVIRONMENT VARIABLES section for those utilities in addition to the global effects described in this section.

LANG This variable shall determine the locale category for native language, local customs, and coded character set in the absence of the *LC_ALL* and other *LC_** (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) environment variables. This can be used by applications to determine the language to use for error messages and instructions, collating sequences, date formats, and so on.

5653	<i>LC_ALL</i>	This variable shall determine the values for all locale categories. The value of the <i>LC_ALL</i> environment variable has precedence over any of the other environment variables starting with <i>LC_</i> (<i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> , <i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i>) and the <i>LANG</i> environment variable.
5654		
5655		
5656		
5657		
5658	<i>LC_COLLATE</i>	This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the <i>strcoll()</i> and <i>strxfrm()</i> functions. Additional semantics of this variable, if any, are implementation-defined.
5659		
5660		
5661		
5662		
5663	<i>LC_CTYPE</i>	This variable shall determine the locale category for character handling functions, such as <i>tolower()</i> , <i>toupper()</i> , and <i>isalpha()</i> . This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.
5664		
5665		
5666		
5667		
5668		
5669		
5670	<i>LC_MESSAGES</i>	This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the <i>catopen()</i>
5671		
5672	XSI	function in determining the message catalog. Additional semantics of this
5673		variable, if any, are implementation-defined. The language and cultural
5674		conventions of diagnostic and informative messages whose format is
5675		unspecified by IEEE Std 1003.1-2001 should be affected by the setting of
5676		<i>LC_MESSAGES</i> .
5677		
5678	<i>LC_MONETARY</i>	This variable shall determine the locale category for monetary-related numeric
5679		formatting information. Additional semantics of this variable, if any, are
5680		implementation-defined.
5681	<i>LC_NUMERIC</i>	This variable shall determine the locale category for numeric formatting (for
5682		example, thousands separator and radix character) information in various
5683		utilities as well as the formatted I/O operations in <i>printf()</i> and <i>scanf()</i> and the
5684		string conversion functions in <i>strtod()</i> . Additional semantics of this variable,
5685		if any, are implementation-defined.
5686	<i>LC_TIME</i>	This variable shall determine the locale category for date and time formatting
5687		information. It affects the behavior of the time functions in <i>strftime()</i> .
5688		Additional semantics of this variable, if any, are implementation-defined.
5689	XSI	<i>NLSPATH</i> This variable shall contain a sequence of templates that the <i>catopen()</i> function
5690		uses when attempting to locate message catalogs. Each template consists of an
5691		optional prefix, one or more conversion specifications, a filename, and an
5692		optional suffix.
5693		For example:
5694		<code>NLSPATH="/system/nlslib/%N.cat"</code>
5695		defines that <i>catopen()</i> should look for all message catalogs in the directory
5696		/system/nlslib , where the catalog name should be constructed from the <i>name</i>
5697		parameter passed to <i>catopen()</i> (<i>%N</i>), with the suffix .cat .
5698		Conversion specifications consist of a ' <i>%</i> ' symbol, followed by a single-letter
5699		keyword. The following keywords are currently defined:

5700	%N	The value of the <i>name</i> parameter passed to <i>catopen()</i> .
5701	%L	The value of the <i>LC_MESSAGES</i> category.
5702	%l	The <i>language</i> element from the <i>LC_MESSAGES</i> category.
5703	%t	The <i>territory</i> element from the <i>LC_MESSAGES</i> category.
5704	%c	The <i>codeset</i> element from the <i>LC_MESSAGES</i> category.
5705	%%	A single ' <i>%</i> ' character.
5706	An empty string is substituted if the specified value is not currently defined.	
5707	The separators underscore (' <i>_</i> ') and period (' <i>.</i> ') are not included in the % <i>t</i>	
5708	and % <i>c</i> conversion specifications.	
5709	Templates defined in <i>NLSPATH</i> are separated by colons (' <i>:</i> '). A leading or	
5710	two adjacent colons " <i>:</i> :" is equivalent to specifying % <i>N</i> . For example:	
5711	<i>NLSPATH</i> =" :% <i>N</i> .cat :/nlslib/% <i>L</i> /% <i>N</i> .cat "	
5712	indicates to <i>catopen()</i> that it should look for the requested message catalog in	
5713	<i>name</i> , <i>name.cat</i> , and <i>/nlslib/category/name.cat</i> , where <i>category</i> is the value of the	
5714	<i>LC_MESSAGES</i> category of the current locale.	
5715	Users should not set the <i>NLSPATH</i> variable unless they have a specific reason	
5716	to override the default system path. Setting <i>NLSPATH</i> to override the default	
5717	system path produces undefined results in the standard utilities and in	
5718	applications with appropriate privileges.	
5719	XSI	The environment variables <i>LANG</i> , <i>LC_ALL</i> , <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5720		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> , and <i>NLSPATH</i> provide for the support of
5721		internationalized applications. The standard utilities shall make use of these environment
5722		variables as described in this section and the individual ENVIRONMENT VARIABLES sections
5723		for the utilities. If these variables specify locale categories that are not based upon the same
5724		underlying codeset, the results are unspecified.
5725	The values of locale categories shall be determined by a precedence order; the first condition met	
5726	below determines the value:	
5727	1. If the <i>LC_ALL</i> environment variable is defined and is not null, the value of <i>LC_ALL</i> shall be	
5728	used.	
5729	2. If the <i>LC_*</i> environment variable (<i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,	
5730	<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i>) is defined and is not null, the value of the	
5731	environment variable shall be used to initialize the category that corresponds to the	
5732	environment variable.	
5733	3. If the <i>LANG</i> environment variable is defined and is not null, the value of the <i>LANG</i>	
5734	environment variable shall be used.	
5735	4. If the <i>LANG</i> environment variable is not set or is set to the empty string, the	
5736	implementation-defined default locale shall be used.	
5737	If the locale value is " <i>C</i> " or " <i>POSIX</i> ", the <i>POSIX</i> locale shall be used and the standard utilities	
5738	behave in accordance with the rules in Section 7.2 (on page 124) for the associated category.	
5739	If the locale value begins with a slash, it shall be interpreted as the pathname of a file that was	
5740	created in the output format used by the <i>localedef</i> utility; see OUTPUT FILES under <i>localedef</i> .	
5741	Referencing such a pathname shall result in that locale being used for the indicated category.	

5742	XSI	If the locale value has the form:
5743		<code>language[_territory] [.codeset]</code>
5744		it refers to an implementation-provided locale, where settings of language, territory, and codeset
5745		are implementation-defined.
5746		<code>LC_COLLATE</code> , <code>LC_CTYPE</code> , <code>LC_MESSAGES</code> , <code>LC_MONETARY</code> , <code>LC_NUMERIC</code> , and <code>LC_TIME</code> are
5747		defined to accept an additional field <code>@modifier</code> , which allows the user to select a specific instance
5748		of localization data within a single category (for example, for selecting the dictionary as opposed
5749		to the character ordering of data). The syntax for these environment variables is thus defined as:
5750		<code>[language[_territory] [.codeset] [@modifier]]</code>
5751		For example, if a user wanted to interact with the system in French, but required to sort German
5752		text files, <code>LANG</code> and <code>LC_COLLATE</code> could be defined as:
5753		<code>LANG=Fr_FR</code>
5754		<code>LC_COLLATE=De_DE</code>
5755		This could be extended to select dictionary collation (say) by use of the <code>@modifier</code> field; for
5756		example:
5757		<code>LC_COLLATE=De_DE@dict</code>
5758		
5759		An implementation may support other formats.
5760		If the locale value is not recognized by the implementation, the behavior is unspecified.
5761		At runtime, these values are bound to a program's locale by calling the <code>setlocale()</code> function.
5762		Additional criteria for determining a valid locale name are implementation-defined.

5763 8.3 Other Environment Variables

5764	<code>COLUMNS</code>	This variable shall represent a decimal integer >0 used to indicate the user's
5765		preferred width in column positions for the terminal screen or window; see
5766		Section 3.103 (on page 49). If this variable is unset or null, the implementation
5767		determines the number of columns, appropriate for the terminal or window,
5768		in an unspecified manner. When <code>COLUMNS</code> is set, any terminal-width
5769		information implied by <code>TERM</code> is overridden. Users and conforming
5770		applications should not set <code>COLUMNS</code> unless they wish to override the
5771		system selection and produce output unrelated to the terminal characteristics.
5772		Users should not need to set this variable in the environment unless there is a
5773		specific reason to override the implementation's default behavior, such as to
5774		display data in an area arbitrarily smaller than the terminal or window.
5775	XSI	<code>DATEMSK</code> Indicates the pathname of the template file used by <code>getdate()</code> .
5776	<code>HOME</code>	The system shall initialize this variable at the time of login to be a pathname of
5777		the user's home directory. See <code><pwd.h></code> .
5778	<code>LINES</code>	This variable shall represent a decimal integer >0 used to indicate the user's
5779		preferred number of lines on a page or the vertical screen or window size in
5780		lines. A line in this case is a vertical measure large enough to hold the tallest
5781		character in the character set being displayed. If this variable is unset or null,
5782		the implementation determines the number of lines, appropriate for the

5783		terminal or window (size, terminal baud rate, and so on), in an unspecified manner. When <i>LINES</i> is set, any terminal-height information implied by <i>TERM</i> is overridden. Users and conforming applications should not set <i>LINES</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
5784		
5785		
5786		
5787		
5788		Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.
5789		
5790		
5791	<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's login name. See <pwd.h> . For a value of <i>LOGNAME</i> to be portable across implementations of IEEE Std 1003.1-2001, the value should be composed of characters from the portable filename character set.
5792		
5793		
5794		
5795	XSI <i>MSGVERB</i>	Describes which message components shall be used in writing messages by <i>fmtmsg()</i> .
5796		
5797	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file known only by a filename. The prefixes shall be separated by a colon (' : '). When a non-zero-length prefix is applied to this filename, a slash shall be inserted between the prefix and the filename. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent colons (" : "), as an initial colon preceding the rest of the list, or as a trailing colon following the rest of the list. A strictly conforming application shall use an actual pathname (such as .) to represent the current working directory in <i>PATH</i> . The list shall be searched from beginning to end, applying the filename to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the pathname being sought contains a slash, the search through the path prefixes shall not be performed. If the pathname begins with a slash, the specified path is resolved (see Section 4.11 (on page 102)). If <i>PATH</i> is unset or is set to null, the path search is implementation-defined.
5798		
5799		
5800		
5801		
5802		
5803		
5804		
5805		
5806		
5807		
5808		
5809		
5810		
5811		
5812		
5813	<i>PWD</i>	This variable shall represent an absolute pathname of the current working directory. It shall not contain any filename components of dot or dot-dot. The value is set by the <i>cd</i> utility.
5814		
5815		
5816	<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in the Shell and Utilities volume of IEEE Std 1003.1-2001, Chapter 2, Shell Command Language, utilities may behave differently from those described in IEEE Std 1003.1-2001.
5817		
5818		
5819		
5820		
5821	<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for programs that need a place to create temporary files.
5822		
5823	<i>TERM</i>	This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.
5824		
5825		
5826		
5827	<i>TZ</i>	This variable shall represent timezone information. The contents of the environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>localtime()</i> , <i>strftime()</i> , <i>mktime()</i> , <i>ctime_r()</i> , and <i>localtime_r()</i> functions, and by various utilities, to override the default timezone. The value of <i>TZ</i> has one of the two
5828		
5829	TSF	
5830		

forms (spaces inserted for clarity):

:characters

or:

std offset dst offset, rule

If *TZ* is of the first format (that is, if the first character is a colon), the characters following the colon are handled in an implementation-defined manner.

The expanded format (for all *TZ*s whose value does not have a colon as the first character) is as follows:

stdoffset [dst [offset] [, start [/time] , end [/time]]]

Where:

std and *dst* Indicate no less than three, nor more than {TZNAME_MAX}, bytes that are the designation for the standard (*std*) or the alternative (*dst*—such as Daylight Savings Time) timezone. Only *std* is required; if *dst* is missing, then the alternative time does not apply in this locale.

Each of these fields may occur in either of two formats quoted or unquoted:

- In the quoted form, the first character shall be the less-than ('<') character and the last character shall be the greater-than ('>') character. All characters between these quoting characters shall be alphanumeric characters from the portable character set in the current locale, the plus-sign ('+') character, or the minus-sign ('-') character. The *std* and *dst* fields in this case shall not include the quoting characters.

- In the unquoted form, all characters in these fields shall be alphabetic characters from the portable character set in the current locale.

The interpretation of these fields is unspecified if either field is less than three bytes (except for the case when *dst* is missing), more than {TZNAME_MAX} bytes, or if they contain characters other than those specified.

offset Indicates the value added to the local time to arrive at Coordinated Universal Time. The *offset* has the form:

hh [: mm [: ss]]

The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*) shall be required and may be a single digit. The *offset* following *std* shall be required. If no *offset* follows *dst*, the alternative time is assumed to be one hour ahead of standard time. One or more digits may be used; the value is always interpreted as a decimal number. The hour shall be between zero and 24, and the minutes (and seconds)—if present—between zero and 59. The result of using values outside of this range is unspecified. If preceded by a '-', the timezone shall be east of the Prime Meridian;

5876		otherwise, it shall be west (which may be indicated by an
5877		optional preceding ' + ').
5878	<i>rule</i>	Indicates when to change to and back from the alternative time.
5879		The <i>rule</i> has the form:
5880		<i>date</i> [/ <i>time</i>] , <i>date</i> [/ <i>time</i>]
5881		where the first <i>date</i> describes when the change from standard to
5882		alternative time occurs and the second <i>date</i> describes when the
5883		change back happens. Each <i>time</i> field describes when, in current
5884		local time, the change to the other time is made.
5885		The format of <i>date</i> is one of the following:
5886	<i>Jn</i>	The Julian day n ($1 \leq n \leq 365$). Leap days shall not be
5887		counted. That is, in all years—including leap years—
5888		February 28 is day 59 and March 1 is day 60. It is
5889		impossible to refer explicitly to the occasional February
5890		29.
5891	<i>n</i>	The zero-based Julian day ($0 \leq n \leq 365$). Leap days shall
5892		be counted, and it is possible to refer to February 29.
5893	<i>Mm.n.d</i>	The d 'th day ($0 \leq d \leq 6$) of week n of month m of the
5894		year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means “the
5895		last d day in month m ” which may occur in either the
5896		fourth or the fifth week). Week 1 is the first week in
5897		which the d 'th day occurs. Day zero is Sunday.
5898		The <i>time</i> has the same format as <i>offset</i> except that no leading sign
5899		(' - ' or ' + ') is allowed. The default, if <i>time</i> is not given, shall be
5900		02:00:00.

Regular Expressions

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings.

Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings, where these character sets are interpreted according to the current locale. While many regular expressions can be interpreted differently depending on the current locale, many features, such as character class expressions, provide for contextual invariance across locales.

The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 171) shall apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in Section 9.4 (on page 175); any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interface in the System Interfaces volume of IEEE Std 1003.1-2001 under *regcomp()*, *regex()*, and related functions.

9.1 Regular Expression Definitions

For the purposes of this section, the following definitions shall apply:

entire regular expression

The concatenated set of one or more BREs or EREs that make up the pattern specified for string selection.

matched

A sequence of zero or more characters shall be said to be matched by a BRE or ERE when the characters in the sequence correspond to a sequence of characters defined by the pattern.

Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. This means that if a character set contains two or more encodings for a graphic symbol, or if the strings searched contain text encoded in more than one codeset, no attempt is made to search for any other representation of the encoded symbol. If that is required, the user can specify equivalence classes containing all variations of the desired graphic symbol.

The search for a matching sequence starts at the beginning of a string and stops when the first sequence matching the expression is found, where “first” is defined to mean “begins earliest in the string”. If the pattern permits a variable number of matching characters and thus there is more than one such sequence starting at that point, the longest such sequence is matched. For example, the BRE `"bb*"` matches the second to fourth characters of the string `"abbbc"`, and the ERE `"(wee|week)(knights|night)"` matches all ten characters of the string `"weeknights"`.

Consistent with the whole match being the longest of the leftmost matches, each subpattern, from left to right, shall match the longest possible string. For this purpose, a null string shall be considered to be longer than no match at all. For example, matching the BRE `"\(.*\)"` against `"abcdef"`, the subexpression `"(\1)"` is `"abcdef"`, and matching the BRE `"\ (a*\)"` against `"bc"`, the subexpression `"(\1)"` is the null string.

When a multi-character collating element in a bracket expression (see Section 9.3.5 (on page 172)) is involved, the longest sequence shall be measured in characters consumed from the string to be matched; that is, the collating element counts not as one element, but as the number of characters it matches.

BRE (ERE) matching a single character

A BRE or ERE that shall match either a single character or a single collating element.

Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5 (on page 172)) can match a collating element.

BRE (ERE) matching multiple characters

A BRE or ERE that shall match a concatenation of single characters or collating elements.

Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE (ERE) special characters.

invalid

This section uses the term “invalid” for certain constructs or conditions. Invalid REs shall cause the utility or function using the RE to generate an error condition. When invalid is not used, violations of the specified syntax or semantics for REs produce undefined results: this may entail an error, enabling an extended syntax for that RE, or using the construct in error as literal characters to be matched. For example, the BRE construct “`\{1,2,3\}`” does not comply with the grammar. A conforming application cannot rely on it producing an error nor matching the literal characters “`\{1,2,3\}`”.

9.2 Regular Expression General Requirements

The requirements in this section shall apply to both basic and extended regular expressions.

The use of regular expressions is generally associated with text processing. REs (BREs and EREs) operate on text strings; that is, zero or more characters followed by an end-of-string delimiter (typically NUL). Some utilities employing regular expressions limit the processing to lines; that is, zero or more characters followed by a <newline>. In the regular expression processing described in IEEE Std 1003.1-2001, the <newline> is regarded as an ordinary character and both a period and a non-matching list can match one. The Shell and Utilities volume of IEEE Std 1003.1-2001 specifies within the individual descriptions of those standard utilities employing regular expressions whether they permit matching of <newline>s; if not stated otherwise, the use of literal <newline>s or any escape sequence equivalent produces undefined results. Those utilities (like *grep*) that do not allow <newline>s to match are responsible for eliminating any <newline> from strings before matching against the RE. The *regcomp()* function in the System Interfaces volume of IEEE Std 1003.1-2001, however, can provide support for such processing without violating the rules of this section.

The interfaces specified in IEEE Std 1003.1-2001 do not permit the inclusion of a NUL character in an RE or in the string to be matched. If during the operation of a standard utility a NUL is included in the text designated to be matched, that NUL may designate the end of the text string for the purposes of matching.

When a standard utility or function that uses regular expressions specifies that pattern matching shall be performed without regard to the case (uppercase or lowercase) of either data or patterns, then when each character in the string is matched against the pattern, not only the character, but also its case counterpart (if any), shall be matched. This definition of case-insensitive processing is intended to allow matching of multi-character collating elements as well as characters, as each character in the string is matched using both its cases. For example, in

5987 a locale where "Ch" is a multi-character collating element and where a matching list expression
 5988 matches such elements, the RE "[.Ch.]" when matched against the string "char" is in
 5989 reality matched against "ch", "Ch", "cH", and "CH".

5990 The implementation shall support any regular expression that does not exceed 256 bytes in
 5991 length.

5992 **9.3 Basic Regular Expressions**

5993 **9.3.1 BREs Matching a Single Character or Collating Element**

5994 A BRE ordinary character, a special character preceded by a backslash, or a period shall match a
 5995 single character. A bracket expression shall match a single character or a single collating
 5996 element.

5997 **9.3.2 BRE Ordinary Characters**

5998 An ordinary character is a BRE that matches itself: any character in the supported character set,
 5999 except for the BRE special characters listed in Section 9.3.3.

6000 The interpretation of an ordinary character preceded by a backslash (' \ ') is undefined, except
 6001 for:

- 6002 • The characters ') ' , ' (' , ' { ' , and ' } '
- 6003 • The digits 1 to 9 inclusive (see Section 9.3.6 (on page 174))
- 6004 • A character inside a bracket expression

6005 **9.3.3 BRE Special Characters**

6006 A BRE special character has special properties in certain contexts. Outside those contexts, or
 6007 when preceded by a backslash, such a character is a BRE that matches the special character itself.
 6008 The BRE special characters and the contexts in which they have their special meaning are as
 6009 follows:

6010 . [\ The period, left-bracket, and backslash shall be special except when used in a bracket
 6011 expression (see Section 9.3.5 (on page 172)). An expression containing a ' [' that is not
 6012 preceded by a backslash and is not part of a bracket expression produces undefined
 6013 results.

6014 * The asterisk shall be special except when used:

- 6015 • In a bracket expression
- 6016 • As the first character of an entire BRE (after an initial ' ^ ', if any)
- 6017 • As the first character of a subexpression (after an initial ' ^ ', if any); see Section
 6018 9.3.6 (on page 174)

6019 ^ The circumflex shall be special when used as:

- 6020 • An anchor (see Section 9.3.8 (on page 175))
- 6021 • The first character of a bracket expression (see Section 9.3.5 (on page 172))

6022 \$ The dollar sign shall be special when used as an anchor.

6023 **9.3.4 Periods in BREs**

6024 A period (' . '), when used outside a bracket expression, is a BRE that shall match any character
 6025 in the supported character set except NUL.

6026 **9.3.5 RE Bracket Expression**

6027 A bracket expression (an expression enclosed in square brackets, "[] ") is an RE that shall match
 6028 a single collating element contained in the non-empty set of collating elements represented by
 6029 the bracket expression.

6030 The following rules and definitions apply to bracket expressions:

6031 1. A bracket expression is either a matching list expression or a non-matching list expression.
 6032 It consists of one or more expressions: collating elements, collating symbols, equivalence
 6033 classes, character classes, or range expressions. The right-bracket ('] ') shall lose its special
 6034 meaning and represent itself in a bracket expression if it occurs first in the list (after an
 6035 initial circumflex (' ^ '), if any). Otherwise, it shall terminate the bracket expression, unless
 6036 it appears in a collating symbol (such as "[.] ") or is the ending right-bracket for a
 6037 collating symbol, equivalence class, or character class. The special characters ' . ', ' * ',
 6038 ' [', and ' \ ' (period, asterisk, left-bracket, and backslash, respectively) shall lose their
 6039 special meaning within a bracket expression.

6040 The character sequences "[. ", "[= ", and "[: " (left-bracket followed by a period, equals-
 6041 sign, or colon) shall be special inside a bracket expression and are used to delimit collating
 6042 symbols, equivalence class expressions, and character class expressions. These symbols
 6043 shall be followed by a valid expression and the matching terminating sequence ".] ",
 6044 "=] ", or ":] ", as described in the following items.

6045 2. A matching list expression specifies a list that shall match any single-character collating
 6046 element in any of the expressions represented in the list. The first character in the list shall
 6047 not be the circumflex; for example, "[abc]" is an RE that matches any of the characters
 6048 'a', 'b', or 'c'. It is unspecified whether a matching list expression matches a multi-
 6049 character collating element that is matched by one of the expressions.

6050 3. A non-matching list expression begins with a circumflex (' ^ '), and specifies a list that
 6051 shall match any single-character collating element except for the expressions represented
 6052 in the list after the leading circumflex. For example, "[^abc]" is an RE that matches any
 6053 character except the characters 'a', 'b', or 'c'. It is unspecified whether a non-matching
 6054 list expression matches a multi-character collating element that is not matched by any of
 6055 the expressions. The circumflex shall have this special meaning only when it occurs first in
 6056 the list, immediately following the left-bracket.

6057 4. A collating symbol is a collating element enclosed within bracket-period ("[. " and ".] ")
 6058 delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page 137).
 6059 Conforming applications shall represent multi-character collating elements as collating
 6060 symbols when it is necessary to distinguish them from a list of the individual characters
 6061 that make up the multi-character collating element. For example, if the string "ch" is a
 6062 collating element defined using the line:

```
6063 collating-element <ch-digraph> from "<c><h>"
```

6064 in the locale definition, the expression "[[.ch.]]" shall be treated as an RE containing
 6065 the collating symbol 'ch', while "[ch]" shall be treated as an RE matching 'c' or 'h'.
 6066 Collating symbols are recognized only inside bracket expressions. If the string is not a
 6067 collating element in the current locale, the expression is invalid.

5. An equivalence class expression shall represent the set of collating elements belonging to an equivalence class, as described in Section 7.3.2.4 (on page 137). Only primary equivalence classes shall be recognized. The class shall be expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal (" [= " and " =] ") delimiters. For example, if 'a', 'à', and 'â' belong to the same equivalence class, then "[[=a=] b]", "[[=à=] b]", and "[[=â=] b]" are each equivalent to "[aââb]". If the collating element does not belong to an equivalence class, the equivalence class expression shall be treated as a collating symbol.

6. A character class expression shall represent the union of two sets:
- a. The set of single-character collating elements whose characters belong to the character class, as defined in the *LC_CTYPE* category in the current locale.
 - b. An unspecified set of multi-character collating elements.

All character classes specified in the current locale shall be recognized. A character class expression is expressed as a character class name enclosed within bracket-colon (" [: " and " :] ") delimiters.

The following character class expressions shall be supported in all locales:

[:alnum:]	[:cntrl:]	[:lower:]	[:space:]
[:alpha:]	[:digit:]	[:print:]	[:upper:]
[:blank:]	[:graph:]	[:punct:]	[:xdigit:]

In addition, character class expressions of the form:

[:name:]

are recognized in those locales where the *name* keyword has been given a **charclass** definition in the *LC_CTYPE* category.

7. In the POSIX locale, a range expression represents the set of collating elements that fall between two elements in the collation sequence, inclusive. In other locales, a range expression has unspecified behavior: strictly conforming applications shall not rely on whether the range expression is valid, or on the set of collating elements matched. A range expression shall be expressed as the starting point and the ending point separated by a hyphen ('-').

In the following, all examples assume the POSIX locale.

The starting range point and the ending range point shall be a collating element or collating symbol. An equivalence class expression used as a starting or ending point of a range expression produces unspecified results. An equivalence class can be used portably within a bracket expression, but only outside the range. If the represented set of collating elements is empty, it is unspecified whether the expression matches nothing, or is treated as invalid.

The interpretation of range expressions where the ending range point is also the starting range point of a subsequent range expression (for example, "[a-m-o]") is undefined.

The hyphen character shall be treated as itself if it occurs first (after an initial '^', if any) or last in the list, or as an ending range point in a range expression. As examples, the expressions "[-ac]" and "[ac -]" are equivalent and match any of the characters 'a', 'c', or '-'; "[^-ac]" and "[^ac -]" are equivalent and match any characters except 'a', 'c', or '-'; the expression "[%- -]" matches any of the characters between '%' and '-' inclusive; the expression "[-@]" matches any of the characters between '-' and '@' inclusive; and the expression "[a--@]" is either invalid or equivalent to '@',

6113 because the letter 'a' follows the symbol '-' in the POSIX locale. To use a hyphen as the
 6114 starting range point, it shall either come first in the bracket expression or be specified as a
 6115 collating symbol; for example, "[[.-]-0]", which matches either a right bracket or
 6116 any character or collating element that collates between hyphen and 0, inclusive.

6117 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the
 6118 '^', if any) and the '-' last within the bracket expression.

6119 9.3.6 BREs Matching Multiple Characters

6120 The following rules can be used to construct BREs matching multiple characters from BREs
 6121 matching a single character:

- 6122 1. The concatenation of BREs shall match the concatenation of the strings matched by each
 6123 component of the BRE.
- 6124 2. A subexpression can be defined within a BRE by enclosing it between the character pairs
 6125 "\(" and "\)". Such a subexpression shall match whatever it would have matched
 6126 without the "\(" and "\)", except that anchoring within subexpressions is optional
 6127 behavior; see Section 9.3.8 (on page 175). Subexpressions can be arbitrarily nested.
- 6128 3. The back-reference expression '\n' shall match the same (possibly empty) string of
 6129 characters as was matched by a subexpression enclosed between "\(" and "\)"
 6130 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the
 6131 *n*th subexpression (the one that begins with the *n*th "\(" from the beginning of the
 6132 pattern and ends with the corresponding paired "\)"). The expression is invalid if less
 6133 than *n* subexpressions precede the '\n'. For example, the expression "\(.*)\1\$" matches a line consisting of two adjacent appearances of the same string, and the expression "\(a\)*\1" fails to match 'a'. When the referenced subexpression matched more than one string, the back-referenced expression shall refer to the last matched string. If the subexpression referenced by the back-reference matches more than one string because of an asterisk ('*') or an interval expression (see item (5)), the back-reference shall match the last (rightmost) of these strings.
- 6140 4. When a BRE matching a single character, a subexpression, or a back-reference is followed
 6141 by the special character asterisk ('*'), together with that asterisk it shall match what zero
 6142 or more consecutive occurrences of the BRE would match. For example, "[ab]*" and
 6143 "[ab][ab]" are equivalent when matching the string "ab".
- 6144 5. When a BRE matching a single character, a subexpression, or a back-reference is followed
 6145 by an interval expression of the format "{m}", "{m,}", or "{m,n}", together
 6146 with that interval expression it shall match what repeated consecutive occurrences of the
 6147 BRE would match. The values of *m* and *n* are decimal integers in the range 0
 6148 $\leq m \leq n \leq \text{RE\_DUP\_MAX}$, where *m* specifies the exact or minimum number of occurrences
 6149 and *n* specifies the maximum number of occurrences. The expression "{m}" shall match
 6150 exactly *m* occurrences of the preceding BRE, "{m,}" shall match at least *m* occurrences,
 6151 and "{m,n}" shall match any number of occurrences between *m* and *n*, inclusive.

6152 For example, in the string "abababcccccd" the BRE "c\{3\}" is matched by
 6153 characters seven to nine, the BRE "\(ab\)\{4,\}" is not matched at all, and the BRE
 6154 "c\{1,3\}d" is matched by characters ten to thirteen.

6155 The behavior of multiple adjacent duplication symbols ('*' and intervals) produces undefined
 6156 results.

6157 A subexpression repeated by an asterisk ('*') or an interval expression shall not match a null
 6158 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or

6159 minimum number of occurrences for the interval expression.

6160 9.3.7 BRE Precedence

6161 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[]
Subexpressions/back-references	\(\) \n
Single-character-BRE duplication	* \{m,n\}
Concatenation	
Anchoring	^ \$

6170 9.3.8 BRE Expression Anchoring

6171 A BRE can be limited to matching strings that begin or end a line; this is called “anchoring”. The
 6172 circumflex and dollar sign special characters shall be considered BRE anchors in the following
 6173 contexts:

- 6174 1. A circumflex ('^') shall be an anchor when used as the first character of an entire BRE.
 6175 The implementation may treat the circumflex as an anchor when used as the first character
 6176 of a subexpression. The circumflex shall anchor the expression (or optionally
 6177 subexpression) to the beginning of a string; only sequences starting at the first character of
 6178 a string shall be matched by the BRE. For example, the BRE "^ab" matches "ab" in the
 6179 string "abcdef", but fails to match in the string "cdefab". The BRE "\(^ab\)" may
 6180 match the former string. A portable BRE shall escape a leading circumflex in a
 6181 subexpression to match a literal circumflex.
- 6182 2. A dollar sign ('\$') shall be an anchor when used as the last character of an entire BRE.
 6183 The implementation may treat a dollar sign as an anchor when used as the last character of
 6184 a subexpression. The dollar sign shall anchor the expression (or optionally subexpression)
 6185 to the end of the string being matched; the dollar sign can be said to match the end-of-
 6186 string following the last character.
- 6187 3. A BRE anchored by both '^' and '\$' shall match only an entire string. For example, the
 6188 BRE "^abcdef\$" matches strings consisting only of "abcdef".

6189 9.4 Extended Regular Expressions

6190 The extended regular expression (ERE) notation and construction rules shall apply to utilities
 6191 defined as using extended regular expressions; any exceptions to the following rules are noted in
 6192 the descriptions of the specific utilities using EREs.

6193 **9.4.1 EREs Matching a Single Character or Collating Element**

6194 An ERE ordinary character, a special character preceded by a backslash, or a period shall match
 6195 a single character. A bracket expression shall match a single character or a single collating
 6196 element. An ERE matching a single character enclosed in parentheses shall match the same as
 6197 the ERE without parentheses would have matched.

6198 **9.4.2 ERE Ordinary Characters**

6199 An ordinary character is an ERE that matches itself. An ordinary character is any character in the
 6200 supported character set, except for the ERE special characters listed in Section 9.4.3. The
 6201 interpretation of an ordinary character preceded by a backslash (' \ ') is undefined.

6202 **9.4.3 ERE Special Characters**

6203 An ERE special character has special properties in certain contexts. Outside those contexts, or
 6204 when preceded by a backslash, such a character shall be an ERE that matches the special
 6205 character itself. The extended regular expression special characters and the contexts in which
 6206 they shall have their special meaning are as follows:

6207 . [\ (The period, left-bracket, backslash, and left-parenthesis shall be special except when
 6208 used in a bracket expression (see Section 9.3.5 (on page 172)). Outside a bracket
 6209 expression, a left-parenthesis immediately followed by a right-parenthesis produces
 6210 undefined results.

6211) The right-parenthesis shall be special when matched with a preceding left-parenthesis,
 6212 both outside a bracket expression.

6213 * + ? { The asterisk, plus-sign, question-mark, and left-brace shall be special except when used
 6214 in a bracket expression (see Section 9.3.5 (on page 172)). Any of the following uses
 6215 produce undefined results:

- 6216 • If these characters appear first in an ERE, or immediately following a vertical-line,
 6217 circumflex, or left-parenthesis
- 6218 • If a left-brace is not part of a valid interval expression (see Section 9.4.6 (on page
 6219 177))

6220 | The vertical-line is special except when used in a bracket expression (see Section 9.3.5
 6221 (on page 172)). A vertical-line appearing first or last in an ERE, or immediately
 6222 following a vertical-line or a left-parenthesis, or immediately preceding a right-
 6223 parenthesis, produces undefined results.

6224 ^ The circumflex shall be special when used as:

- 6225 • An anchor (see Section 9.4.9 (on page 178))
- 6226 • The first character of a bracket expression (see Section 9.3.5 (on page 172))

6227 \$ The dollar sign shall be special when used as an anchor.

6228 9.4.4 Periods in EREs

6229 A period ('.'), when used outside a bracket expression, is an ERE that shall match any
6230 character in the supported character set except NUL.

6231 9.4.5 ERE Bracket Expression

6232 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see Section
6233 9.3.5 (on page 172).

6234 9.4.6 EREs Matching Multiple Characters

6235 The following rules shall be used to construct EREs matching multiple characters from EREs
6236 matching a single character:

- 6237 1. A concatenation of EREs shall match the concatenation of the character sequences matched
6238 by each component of the ERE. A concatenation of EREs enclosed in parentheses shall
6239 match whatever the concatenation without the parentheses matches. For example, both the
6240 ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of the string
6241 "abcdefabcdef".
- 6242 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6243 the special character plus-sign ('+'), together with that plus-sign it shall match what one
6244 or more consecutive occurrences of the ERE would match. For example, the ERE
6245 "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde". And,
6246 "[ab]+" and "[ab][ab]*" are equivalent.
- 6247 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6248 the special character asterisk ('*'), together with that asterisk it shall match what zero or
6249 more consecutive occurrences of the ERE would match. For example, the ERE "b*c"
6250 matches the first character in the string "cabbbbcde", and the ERE "b*cd" matches the
6251 third to seventh characters in the string "cabbbbcdebbbbbbcbdbc". And, "[ab]*" and
6252 "[ab][ab]" are equivalent when matching the string "ab".
- 6253 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6254 the special character question-mark ('?'), together with that question-mark it shall match
6255 what zero or one consecutive occurrences of the ERE would match. For example, the ERE
6256 "b?c" matches the second character in the string "acabbbbcde".
- 6257 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6258 an interval expression of the format "{m}", "{m,}", or "{m,n}", together with that
6259 interval expression it shall match what repeated consecutive occurrences of the ERE would
6260 match. The values of m and n are decimal integers in the range $0 \leq m \leq n \leq \{RE\_DUP\_MAX\}$,
6261 where m specifies the exact or minimum number of occurrences and n specifies the
6262 maximum number of occurrences. The expression "{m}" matches exactly m occurrences
6263 of the preceding ERE, "{m,}" matches at least m occurrences, and "{m,n}" matches any
6264 number of occurrences between m and n , inclusive.

6265 For example, in the string "abababcccccd" the ERE "c{3}" is matched by characters
6266 seven to nine and the ERE "(ab){2,}" is matched by characters one to six.

6267 The behavior of multiple adjacent duplication symbols ('+', '*', '?', and intervals) produces
6268 undefined results.

6269 An ERE matching a single character repeated by an '*', '?', or an interval expression shall not
6270 match a null expression unless this is the only match for the repetition or it is necessary to satisfy
6271 the exact or minimum number of occurrences for the interval expression.

6272 9.4.7 ERE Alternation

6273 Two EREs separated by the special character vertical-line ('|') shall match a string that is
 6274 matched by either. For example, the ERE "a (bc) | d" matches the string "abc" and the string
 6275 "ad". Single characters, or expressions matching single characters, separated by the vertical bar
 6276 and enclosed in parentheses, shall be treated as an ERE matching a single character.

6277 9.4.8 ERE Precedence

6278 The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[]
Grouping	()
Single-character-ERE duplication	* + ? {m,n}
Concatenation	
Anchoring	^ \$
Alternation	

6288 For example, the ERE "abba|cde" matches either the string "abba" or the string "cde"
 6289 (rather than the string "abbade" or "abbcde", because concatenation has a higher order of
 6290 precedence than alternation).

6291 9.4.9 ERE Expression Anchoring

6292 An ERE can be limited to matching strings that begin or end a line; this is called "anchoring".
 6293 The circumflex and dollar sign special characters shall be considered ERE anchors when used
 6294 anywhere outside a bracket expression. This shall have the following effects:

- 6295 1. A circumflex ('^') outside a bracket expression shall anchor the expression or
 6296 subexpression it begins to the beginning of a string; such an expression or subexpression
 6297 can match only a sequence starting at the first character of a string. For example, the EREs
 6298 "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string
 6299 "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents the
 6300 expression "^b" from matching starting at the first character.
- 6301 2. A dollar sign ('\$') outside a bracket expression shall anchor the expression or
 6302 subexpression it ends to the end of a string; such an expression or subexpression can
 6303 match only a sequence ending at the last character of a string. For example, the EREs
 6304 "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string
 6305 "cdefab", and the ERE "e\$f" is valid, but can never match because the 'f' prevents the
 6306 expression "e\$" from matching ending at the last character.

6307 9.5 Regular Expression Grammar

6308 Grammars describing the syntax of both basic and extended regular expressions are presented in
6309 this section. The grammar takes precedence over the text. See the Shell and Utilities volume of
6310 IEEE Std 1003.1-2001, Section 1.10, Grammar Conventions.

6311 9.5.1 BRE/ERE Grammar Lexical Conventions

6312 The lexical conventions for regular expressions are as described in this section.

6313 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6314 The following tokens are processed (in addition to those string constants shown in the
6315 grammar):

6316 **COLL_ELEM_SINGLE**

6317 Any single-character collating element, unless it is a **META_CHAR**.

6318 **COLL_ELEM_MULTI**

6319 Any multi-character collating element.

6320 **BACKREF**

6321 Applicable only to basic regular expressions. The character string
consisting of '****' followed by a single-digit numeral, '1' to '9'.

6322 **DUP_COUNT**

6323 Represents a numeric constant. It shall be an integer in the range 0
6324 $\leq \mathbf{DUP\_COUNT} \leq \{\mathbf{RE\_DUP\_MAX}\}$. This token is only recognized when
6325 the context of the grammar requires it. At all other times, digits not
preceded by '****' are treated as **ORD_CHAR**.

6326 **META_CHAR**

One of the characters:

6327 **^** When found first in a bracket expression

6328 **-** When found anywhere but first (after an initial '**^**', if any) or
6329 last in a bracket expression, or as the ending range point in a
6330 range expression

6331 **]** When found anywhere but first (after an initial '**^**', if any) in a
6332 bracket expression

6333 **L_ANCHOR**

6334 Applicable only to basic regular expressions. The character '**^**' when it
6335 appears as the first character of a basic regular expression and when not
6336 **QUOTED_CHAR**. The '**^**' may be recognized as an anchor elsewhere;
see Section 9.3.8 (on page 175).

6337 **ORD_CHAR**

A character, other than one of the special characters in **SPEC_CHAR**.

6338 **QUOTED_CHAR**

In a BRE, one of the character sequences:

6339 **\^** **\.** ***** **\[** **\\$** ****

6340 In an ERE, one of the character sequences:

6341 **\^** **\.** **\[** **\\$** **\(** **\)** ****

6342 ***** **\+** **\?** **\{** ****

6343 **R_ANCHOR**

6344 (Applicable only to basic regular expressions.) The character '**\$**' when it
6345 appears as the last character of a basic regular expression and when not
6346 **QUOTED_CHAR**. The '**\$**' may be recognized as an anchor elsewhere;
see Section 9.3.8 (on page 175).

6347 **SPEC_CHAR** For basic regular expressions, one of the following special characters:

6348 . Anywhere outside bracket expressions

6349 \ Anywhere outside bracket expressions

6350 [Anywhere outside bracket expressions

6351 ^ When used as an anchor (see Section 9.3.8 (on page 175)) or

6352 when first in a bracket expression

6353 \$ When used as an anchor

6354 * Anywhere except first in an entire RE, anywhere in a bracket

6355 expression, directly following "\(", directly following an

6356 anchoring '^'

6357 For extended regular expressions, shall be one of the following special

6358 characters found anywhere outside bracket expressions:

6359 ^ . [\$ () |

6360 * + ? { \

6361 (The close-parenthesis shall be considered special in this context only if

6362 matched with a preceding open-parenthesis.)

6363 9.5.2 RE and Bracket Expression Grammar

6364 This section presents the grammar for basic regular expressions, including the bracket

6365 expression grammar that is common to both BREs and EREs.

6366 %token ORD_CHAR QUOTED_CHAR DUP_COUNT

6367 %token BACKREF L_ANCHOR R_ANCHOR

6368 %token Back_open_paren Back_close_paren

6369 /* '\(' '\)' */

6370 %token Back_open_brace Back_close_brace

6371 /* '\{' '\}' */

6372 /* The following tokens are for the Bracket Expression

6373 grammar common to both REs and EREs. */

6374 %token COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR

6375 %token Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close

6376 /* '[' '=' '[' '.' ']' ':' ']' */

6377 %token class_name

6378 /* class_name is a keyword to the LC_CTYPE locale category */

6379 /* (representing a character class) in the current locale */

6380 /* and is only recognized between [: and :] */

6381 %start basic_reg_exp

6382 %%

6383 /* -----

6384 Basic Regular Expression

6385 -----

6386 */

6387 basic_reg_exp : RE_expression

```

6388          | L_ANCHOR
6389          |                                     R_ANCHOR
6390          | L_ANCHOR                             R_ANCHOR
6391          | L_ANCHOR RE_expression
6392          |                                     RE_expression R_ANCHOR
6393          | L_ANCHOR RE_expression R_ANCHOR
6394          ;
6395 RE_expression : simple_RE
6396          | RE_expression simple_RE
6397          ;
6398 simple_RE : nondupl_RE
6399          | nondupl_RE RE_dupl_symbol
6400          ;
6401 nondupl_RE : one_char_or_coll_elem_RE
6402          | Back_open_paren RE_expression Back_close_paren
6403          | BACKREF
6404          ;
6405 one_char_or_coll_elem_RE : ORD_CHAR
6406          | QUOTED_CHAR
6407          | '.'
6408          | bracket_expression
6409          ;
6410 RE_dupl_symbol : '*'
6411          | Back_open_brace DUP_COUNT Back_close_brace
6412          | Back_open_brace DUP_COUNT ',' Back_close_brace
6413          | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6414          ;
6415 /* -----
6416    Bracket Expression
6417    -----
6418 */
6419 bracket_expression : '[' matching_list ']'
6420          | '[' nonmatching_list ']'
6421          ;
6422 matching_list : bracket_list
6423          ;
6424 nonmatching_list : '^' bracket_list
6425          ;
6426 bracket_list : follow_list
6427          | follow_list '-'
6428          ;
6429 follow_list : expression_term
6430          | follow_list expression_term
6431          ;
6432 expression_term : single_expression
6433          | range_expression
6434          ;
6435 single_expression : end_range
6436          | character_class
6437          | equivalence_class
6438          ;
6439 range_expression : start_range end_range

```

```

6440         | start_range '-'
6441         ;
6442     start_range      : end_range '-'
6443     ;
6444     end_range        : COLL_ELEM_SINGLE
6445         | collating_symbol
6446     ;
6447     collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
6448         | Open_dot COLL_ELEM_MULTI Dot_close
6449         | Open_dot META_CHAR Dot_close
6450     ;
6451     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6452         | Open_equal COLL_ELEM_MULTI Equal_close
6453     ;
6454     character_class : Open_colon class_name Colon_close
6455     ;

```

6456 The BRE grammar does not permit **L_ANCHOR** or **R_ANCHOR** inside "\(" and "\)" (which
6457 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the
6458 application, as noted in Section 9.3.8 (on page 175). Implementations are permitted to extend the
6459 language to interpret '^' and '\$' as anchors in these locations, and as such, conforming
6460 applications cannot use unescaped '^' and '\$' in positions inside "\(" and "\)" that might
6461 be interpreted as anchors.

6462 9.5.3 ERE Grammar

6463 This section presents the grammar for extended regular expressions, excluding the bracket
6464 expression grammar.

6465 **Note:** The bracket expression grammar and the associated **%token** lines are identical between BREs
6466 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6467 %token  ORD_CHAR QUOTED_CHAR DUP_COUNT
6468 %start  extended_reg_exp
6469 %%
6470 /* -----
6471    Extended Regular Expression
6472    -----
6473 */
6474 extended_reg_exp      :                ERE_branch
6475         | extended_reg_exp '|' ERE_branch
6476     ;
6477 ERE_branch            :                ERE_expression
6478         | ERE_branch ERE_expression
6479     ;
6480 ERE_expression        : one_char_or_coll_elem_ERE
6481         | '^'
6482         | '$'
6483         | '(' extended_reg_exp ')'
6484         | ERE_expression ERE_dupl_symbol
6485     ;
6486 one_char_or_coll_elem_ERE : ORD_CHAR
6487         | QUOTED_CHAR
6488         | '.'

```

```

6489             | bracket_expression
6490             ;
6491 ERE_dupl_symbol : '*'
6492             | '+'
6493             | '?'
6494             | '{' DUP_COUNT '}'
6495             | '{' DUP_COUNT ',' '}'
6496             | '{' DUP_COUNT ',' DUP_COUNT '}'
6497             ;

```

6498 The ERE grammar does not permit several constructs that previous sections specify as having
 6499 undefined results:

- 6500 • **ORD_CHAR** preceded by '`\`'
- 6501 • One or more *ERE_dupl_symbols* appearing first in an ERE, or immediately following '`|`',
 6502 '`^`', or '`(`'
- 6503 • '`{`' not part of a valid *ERE_dupl_symbol*
- 6504 • '`|`' appearing first or last in an ERE, or immediately following '`|`' or '`(`', or immediately
 6505 preceding '`)`'

6506 Implementations are permitted to extend the language to allow these. Conforming applications
 6507 cannot use such constructs.

Directory Structure and Devices

6509

10.1 Directory Structure and Files

The following directories shall exist on conforming systems and conforming applications shall make use of them only as described. Strictly conforming applications shall not assume the ability to create files in any of these directories, unless specified below.

/ The root directory.

/dev Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

The following directory shall exist on conforming systems and shall be used as described:

/tmp A directory made available for applications that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable:

/dev/null An infinite data source and data sink. Data written to **/dev/null** shall be discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

/dev/tty In each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs or shell procedures that wish to be sure of writing messages to or reading data from the terminal no matter how output has been redirected. It can also be used for applications that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

The following file shall exist on conforming systems and need not be readable or writable:

/dev/console The **/dev/console** file is a generic name given to the system console (see Section 3.382 (on page 88)). It is usually linked to an implementation-defined special file. It shall provide an interface to the system console conforming to the requirements of the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.

10.2 Output Devices and Terminal Types

The utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 historically have been implemented on a wide range of terminal types, but a conforming implementation need not support all features of all utilities on every conceivable terminal. IEEE Std 1003.1-2001 states which features are optional for certain classes of terminals in the individual utility description sections. The implementation shall document which terminal types it supports and which of these features and utilities are not supported by each terminal.

When a feature or utility is not supported on a specific terminal type, as allowed by IEEE Std 1003.1-2001, and the implementation considers such a condition to be an error preventing use of the feature or utility, the implementation shall indicate such conditions through diagnostic messages or exit status values or both (as appropriate to the specific utility description) that inform the user that the terminal type lacks the appropriate capability.

IEEE Std 1003.1-2001 uses a notational convention based on historical practice that identifies some of the control characters defined in Section 7.3.1 (on page 126) in a manner easily remembered by users on many terminals. The correspondence between this “<control>-char” notation and the actual control characters is shown in the following table. When IEEE Std 1003.1-2001 refers to a character by its <control>-name, it is referring to the actual control character shown in the Value column of the table, which is not necessarily the exact control key sequence on all terminals. Some terminals have keyboards that do not allow the direct transmission of all the non-alphanumeric characters shown. In such cases, the system documentation shall describe which data sequences transmitted by the terminal are interpreted by the system as representing the special characters.

Table 10-1 Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y		
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[	<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\ <control>-]	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-^	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-_	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-?	<US>	<US>
<control>-P	<DLE>	<DLE>		<DEL>	<DEL>

Note: The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that the keystrokes represent control-shift-letter sequences.

General Terminal Interface

This chapter describes a general terminal interface that shall be provided. It shall be supported on any asynchronous communications ports if the implementation provides them. It is implementation-defined whether it supports network connections or synchronous ports, or both.

11.1 Interface Characteristics

11.1.1 Opening a Terminal Device File

When a terminal device file is opened, it normally causes the thread to wait until a connection is established. In practice, application programs seldom open these files; they are opened by special programs and become an application's standard input, output, and error files.

As described in *open()*, opening a terminal device file with the *O_NONBLOCK* flag clear shall cause the thread to block until the terminal device is ready and available. If *CLOCAL* mode is not set, this means blocking until a connection is established. If *CLOCAL* mode is set in the terminal, or the *O_NONBLOCK* flag is specified in the *open()*, the *open()* function shall return a file descriptor without waiting for a connection to be established.

11.1.2 Process Groups

A terminal may have a foreground process group associated with it. This foreground process group plays a special role in handling signal-generating input characters, as discussed in Section 11.1.9 (on page 191).

A command interpreter process supporting job control can allocate the terminal to different jobs, or process groups, by placing related processes in a single process group and associating this process group with the terminal. A terminal's foreground process group may be set or examined by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*. The terminal interface aids in this allocation by restricting access to the terminal by processes that are not in the current process group; see Section 11.1.4 (on page 188).

When there is no longer any process whose process ID or process group ID matches the foreground process group ID, the terminal shall have no foreground process group. It is unspecified whether the terminal has a foreground process group when there is a process whose process ID matches the foreground process group ID, but whose process group ID does not. No actions defined in IEEE Std 1003.1-2001, other than allocation of a controlling terminal or a successful call to *tcsetpgrp()*, shall cause a process group to become the foreground process group of the terminal.

6609 11.1.3 The Controlling Terminal

6610 A terminal may belong to a process as its controlling terminal. Each process of a session that has
 6611 a controlling terminal has the same controlling terminal. A terminal may be the controlling
 6612 terminal for at most one session. The controlling terminal for a session is allocated by the session
 6613 leader in an implementation-defined manner. If a session leader has no controlling terminal, and
 6614 opens a terminal device file that is not already associated with a session without using the
 6615 O_NOCTTY option (see *open()*), it is implementation-defined whether the terminal becomes the
 6616 controlling terminal of the session leader. If a process which is not a session leader opens a
 6617 terminal file, or the O_NOCTTY option is used on *open()*, then that terminal shall not become
 6618 the controlling terminal of the calling process. When a controlling terminal becomes associated
 6619 with a session, its foreground process group shall be set to the process group of the session
 6620 leader.

6621 The controlling terminal is inherited by a child process during a *fork()* function call. A process
 6622 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;
 6623 other processes remaining in the old session that had this terminal as their controlling terminal
 6624 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in
 6625 the current session) associated with the controlling terminal, it is unspecified whether all
 6626 processes that had that terminal as their controlling terminal cease to have any controlling
 6627 terminal. Whether and how a session leader can reacquire a controlling terminal after the
 6628 controlling terminal has been relinquished in this fashion is unspecified. A process does not
 6629 relinquish its controlling terminal simply by closing all of its file descriptors associated with the
 6630 controlling terminal if other processes continue to have it open.

6631 When a controlling process terminates, the controlling terminal is dissociated from the current
 6632 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by
 6633 other processes in the earlier session may be denied, with attempts to access the terminal treated
 6634 as if a modem disconnect had been sensed.

6635 11.1.4 Terminal Access Control

6636 If a process is in the foreground process group of its controlling terminal, read operations shall
 6637 be allowed, as described in Section 11.1.5 (on page 189). Any attempts by a process in a
 6638 background process group to read from its controlling terminal cause its process group to be
 6639 sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is
 6640 ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is
 6641 orphaned, the *read()* shall return *-1*, with *errno* set to [EIO] and no signal shall be sent. The
 6642 default action of the SIGTTIN signal shall be to stop the process to which it is sent. See
 6643 **<signal.h>**.

6644 If a process is in the foreground process group of its controlling terminal, write operations shall
 6645 be allowed as described in Section 11.1.8 (on page 191). Attempts by a process in a background
 6646 process group to write to its controlling terminal shall cause the process group to be sent a
 6647 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if
 6648 TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is
 6649 allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the
 6650 process group of the writing process is orphaned, and the writing process is not ignoring or
 6651 blocking the SIGTTOU signal, the *write()* shall return *-1*, with *errno* set to [EIO] and no signal
 6652 shall be sent.

6653 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that
 6654 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set
 6655 (see Section 11.2.5 (on page 197), *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()*, *tcsetattr()*, and
 6656 *tcsetpgrp()*).

6657 11.1.5 Input Processing and Reading Data

6658 A terminal device associated with a terminal device file may operate in full-duplex mode, so that
 6659 data may arrive even while output is occurring. Each terminal device file has an input queue
 6660 associated with it, into which incoming data is stored by the system before being read by a
 6661 process. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be
 6662 stored in the input queue. The behavior of the system when this limit is exceeded is
 6663 implementation-defined.

6664 Two general kinds of input processing are available, determined by whether the terminal device
 6665 file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 and
 6666 Section 11.1.7 (on page 190). Additionally, input characters are processed according to the *c_iflag*
 6667 (see Section 11.2.2 (on page 193)) and *c_lflag* (see Section 11.2.5 (on page 197)) fields. Such
 6668 processing can include “echoing”, which in general means transmitting input characters
 6669 immediately back to the terminal when they are received from the terminal. This is useful for
 6670 terminals that can operate in full-duplex mode.

6671 The manner in which data is provided to a process reading from a terminal device file is
 6672 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether
 6673 or not the O_NONBLOCK flag is set by *open()* or *fcntl()*.

6674 If the O_NONBLOCK flag is clear, then the read request shall be blocked until data is available
 6675 or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall be
 6676 completed, without blocking, in one of three ways:

- 6677 1. If there is enough data available to satisfy the entire request, the *read()* shall complete
 6678 successfully and shall return the number of bytes read.
- 6679 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete
 6680 successfully, having read as much data as possible, and shall return the number of bytes it
 6681 was able to read.
- 6682 3. If there is no data available, the *read()* shall return -1, with *errno* set to [EAGAIN].

6683 When data is available depends on whether the input processing mode is canonical or non-
 6684 canonical. Section 11.1.6 and Section 11.1.7 (on page 190) describe each of these input processing
 6685 modes.

6686 11.1.6 Canonical Mode Input Processing

6687 In canonical mode input processing, terminal input is processed in units of lines. A line is
 6688 delimited by a newline character (NL), an end-of-file character (EOF), or an end-of-line (EOL)
 6689 character. See Section 11.1.9 (on page 191) for more information on EOF and EOL. This means
 6690 that a read request shall not return until an entire line has been typed or a signal has been
 6691 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall
 6692 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even
 6693 one, may be requested in a *read()* without losing information.

6694 If {MAX_CANON} is defined for this terminal device, it shall be a limit on the number of bytes
 6695 in a line. The behavior of the system when this limit is exceeded is implementation-defined. If
 6696 {MAX_CANON} is not defined, there shall be no such limit; see *pathconf()*.

6697 Erase and kill processing occur when either of two special characters, the ERASE and KILL
 6698 characters (see Section 11.1.9 (on page 191)), is received. This processing shall affect data in the
 6699 input queue that has not yet been delimited by an NL, EOF, or EOL character. This un-delimited
 6700 data makes up the current line. The ERASE character shall delete the last character in the current
 6701 line, if there is one. The KILL character shall delete all data in the current line, if there is any. The
 6702 ERASE and KILL characters shall have no effect if there is no data in the current line. The ERASE

6703 and KILL characters themselves shall not be placed in the input queue.

6704 **11.1.7 Non-Canonical Mode Input Processing**

6705 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and
6706 kill processing shall not occur. The values of the MIN and TIME members of the *c_cc* array are
6707 used to determine how to process the bytes received. IEEE Std 1003.1-2001 does not specify
6708 whether the setting of O_NONBLOCK takes precedence over MIN or TIME settings. Therefore,
6709 if O_NONBLOCK is set, *read()* may return immediately, regardless of the setting of MIN or
6710 TIME. Also, if no data is available, *read()* may either return 0, or return -1 with *errno* set to
6711 [EAGAIN].

6712 MIN represents the minimum number of bytes that should be received when the *read()* function
6713 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and
6714 short-term data transmissions. If MIN is greater than {MAX_INPUT}, the response to the request
6715 is undefined. The four possible values for MIN and TIME and their interactions are described
6716 below.

6717 **Case A: MIN>0, TIME>0**

6718 In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is
6719 received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction
6720 between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall
6721 be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer
6722 is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN
6723 bytes are received, the characters received to that point shall be returned to the user. Note that if
6724 TIME expires at least one byte shall be returned because the timer would not have been enabled
6725 unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and
6726 TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in
6727 the buffer at the time of the *read()*, the result shall be as if data has been received immediately
6728 after the *read()*.

6729 **Case B: MIN>0, TIME=0**

6730 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A
6731 pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall
6732 block until MIN bytes are received), or a signal is received. A program that uses case B to read
6733 record-based terminal I/O may block indefinitely in the read operation.

6734 **Case C: MIN=0, TIME>0**

6735 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read
6736 timer that shall be activated as soon as the *read()* function is processed. A read shall be satisfied
6737 as soon as a single byte is received or the read timer expires. Note that in case C if the timer
6738 expires, no bytes shall be returned. If the timer does not expire, the only way the read can be
6739 satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely
6740 waiting for a byte; if no byte is received within TIME*0.1 seconds after the read is initiated, the
6741 *read()* shall return a value of zero, having read no data. If data is in the buffer at the time of the
6742 *read()*, the timer shall be started as if data has been received immediately after the *read()*.

6743 **Case D: MIN=0, TIME=0**

6744 The minimum of either the number of bytes requested or the number of bytes currently available
 6745 shall be returned without waiting for more bytes to be input. If no characters are available, *read()*
 6746 shall return a value of zero, having read no data.

6747 **11.1.8 Writing Data and Output Processing**

6748 When a process writes one or more bytes to a terminal device file, they are processed according
 6749 to the *c_oflag* field (see Section 11.2.3 (on page 194)). The implementation may provide a
 6750 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have
 6751 been scheduled for transmission to the device, but the transmission has not necessarily
 6752 completed. See *write()* for the effects of O_NONBLOCK on *write()*.

6753 **11.1.9 Special Characters**

6754 Certain characters have special functions on input or output or both. These functions are
 6755 summarized as follows:

6756 **INTR** Special character on input, which is recognized if the ISIG flag is set. Generates a
 6757 SIGINT signal which is sent to all processes in the foreground process group for which
 6758 the terminal is the controlling terminal. If ISIG is set, the INTR character shall be
 6759 discarded when processed.

6760 **QUIT** Special character on input, which is recognized if the ISIG flag is set. Generates a
 6761 SIGQUIT signal which is sent to all processes in the foreground process group for
 6762 which the terminal is the controlling terminal. If ISIG is set, the QUIT character shall be
 6763 discarded when processed.

6764 **ERASE** Special character on input, which is recognized if the ICANON flag is set. Erases the
 6765 last character in the current line; see Section 11.1.6 (on page 189). It shall not erase
 6766 beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is
 6767 set, the ERASE character shall be discarded when processed.

6768 **KILL** Special character on input, which is recognized if the ICANON flag is set. Deletes the
 6769 entire line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the KILL
 6770 character shall be discarded when processed.

6771 **EOF** Special character on input, which is recognized if the ICANON flag is set. When
 6772 received, all the bytes waiting to be read are immediately passed to the process without
 6773 waiting for a newline, and the EOF is discarded. Thus, if there are no bytes waiting
 6774 (that is, the EOF occurred at the beginning of a line), a byte count of zero shall be
 6775 returned from the *read()*, representing an end-of-file indication. If ICANON is set, the
 6776 EOF character shall be discarded when processed.

6777 **NL** Special character on input, which is recognized if the ICANON flag is set. It is the line
 6778 delimiter newline. It cannot be changed.

6779 **EOL** Special character on input, which is recognized if the ICANON flag is set. It is an
 6780 additional line delimiter, like NL.

6781 **SUSP** If the ISIG flag is set, receipt of the SUSP character shall cause a SIGTSTP signal to be
 6782 sent to all processes in the foreground process group for which the terminal is the
 6783 controlling terminal, and the SUSP character shall be discarded when processed.

6784 **STOP** Special character on both input and output, which is recognized if the IXON (output
 6785 control) or IXOFF (input control) flag is set. Can be used to suspend output
 6786 temporarily. It is useful with CRT terminals to prevent output from disappearing

6787 before it can be read. If IXON is set, the STOP character shall be discarded when
 6788 processed.

6789 **START** Special character on both input and output, which is recognized if the IXON (output
 6790 control) or IXOFF (input control) flag is set. Can be used to resume output that has
 6791 been suspended by a STOP character. If IXON is set, the START character shall be
 6792 discarded when processed.

6793 **CR** Special character on input, which is recognized if the ICANON flag is set; it is the
 6794 carriage-return character. When ICANON and ICRNL are set and IGNCR is not set,
 6795 this character shall be translated into an NL, and shall have the same effect as an NL
 6796 character.

6797 The NL and CR characters cannot be changed. It is implementation-defined whether the START
 6798 and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and
 6799 SUSP shall be changeable to suit individual tastes. Special character functions associated with
 6800 changeable special control characters can be disabled individually.

6801 If two or more special characters have the same value, the function performed when that
 6802 character is received is undefined.

6803 A special character is recognized not only by its value, but also by its context; for example, an
 6804 implementation may support multi-byte sequences that have a meaning different from the
 6805 meaning of the bytes when considered individually. Implementations may also support
 6806 additional single-byte functions. These implementation-defined multi-byte or single-byte
 6807 functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without
 6808 interpretation, except as required to recognize the special characters defined in this section.

6809 **XSI** If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding ‘\’
 6810 character, in which case no special function shall occur.

6811 **11.1.10 Modem Disconnect**

6812 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if
 6813 CLOCAL is not set in the *c_cflag* field for the terminal (see Section 11.2.4 (on page 196)), the
 6814 SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling
 6815 terminal. Unless other arrangements have been made, this shall cause the controlling process to
 6816 terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of
 6817 zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-
 6818 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()*
 6819 also exists, it is unspecified whether on EOF condition or [EIO] is returned. Any subsequent
 6820 *write()* to the terminal device shall return -1, with *errno* set to [EIO], until the device is closed.

6821 **11.1.11 Closing a Terminal Device File**

6822 The last process to close a terminal device file shall cause any output to be sent to the device and
 6823 any input to be discarded. If HUPCL is set in the control modes and the communications port
 6824 supports a disconnect function, the terminal device shall perform a disconnect.

11.2 Parameters that Can be Set

11.2.1 The `termios` Structure

Routines that need to control certain terminal I/O characteristics shall do so by using the **termios** structure as defined in the `<termios.h>` header. The members of this structure include (but are not limited to):

Member Type	Array Size	Member Name	Description
tcflag_t	NCCS	<i>c_iflag</i>	Input modes.
tcflag_t		<i>c_oflag</i>	Output modes.
tcflag_t		<i>c_cflag</i>	Control modes.
tcflag_t		<i>c_lflag</i>	Local modes.
cc_t		<i>c_cc[]</i>	Control characters.

The types **tcflag_t** and **cc_t** are defined in the `<termios.h>` header. They shall be unsigned integer types.

11.2.2 Input Modes

Values of the *c_iflag* field describe the basic terminal input control, and are composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in `<termios.h>`:

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

In the context of asynchronous serial data transmission, a break condition shall be defined as a sequence of zero-valued bits that continues for more than the time to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a time equivalent to more than one byte. In contexts other than asynchronous serial data transmission, the definition of a break condition is implementation-defined.

If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition shall flush the input and output queues, and if the terminal is the controlling terminal of a foreground process group, the break condition shall generate a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

6869 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than
 6870 break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is
 6871 a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid
 6872 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff
 6873 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be
 6874 given to the application as a single byte 0x00.

6875 If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking
 6876 shall be disabled, allowing output parity generation without input parity errors. Note that
 6877 whether input parity checking is enabled or disabled is independent of whether parity detection
 6878 is enabled or disabled (see Section 11.2.4 (on page 196)). If parity detection is enabled but input
 6879 parity checking is disabled, the hardware to which the terminal is connected shall recognize the
 6880 parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

6881 If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits
 6882 shall be processed.

6883 If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a
 6884 received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a
 6885 received CR character shall be translated into an NL character.

6886 XSI If IXANY is set, any input character shall restart output that has been suspended.

6887 If IXON is set, start/stop output control shall be enabled. A received STOP character shall
 6888 suspend output and a received START character shall restart output. When IXON is set, START
 6889 and STOP characters are not read, but merely perform flow control functions. When IXON is not
 6890 set, the START and STOP characters shall be read.

6891 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP
 6892 characters, which are intended to cause the terminal device to stop transmitting data, as needed
 6893 to prevent the input queue from overflowing and causing implementation-defined behavior, and
 6894 shall transmit START characters, which are intended to cause the terminal device to resume
 6895 transmitting data, as soon as the device can continue transmitting data without risk of
 6896 overflowing the input queue. The precise conditions under which STOP and START characters
 6897 are transmitted are implementation-defined.

6898 The initial input control value after *open()* is implementation-defined.

6899 11.2.3 Output Modes

6900 The *c_oflag* field specifies the terminal interface's treatment of output, and is composed of the
 6901 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
 6902 symbols in the following table are defined in **<termios.h>**:

6903
6904
6905
6906 XSI
6907
6908
6909
6910
6911
6912
6913
6914
6915
6916
6917
6918
6919
6920
6921
6922
6923
6924
6925
6926
6927
6928
6929
6930
6931
6932
6933

Mask Name	Description
OPOST	Perform output processing.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

6934
6935
6936

If OPOST is set, output data shall be post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters shall be transmitted without change.

6937 XSI
6938
6939
6940
6941
6942
6943

If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the CR character is actually transmitted.

6944
6945
6946
6947
6948

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character shall be DEL; otherwise, NUL.

6949

If a form-feed or vertical-tab delay is specified, it shall last for about 2 seconds.

6950
6951

Newline delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall transmit two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be about 0.10 seconds. Type 3 specifies that tabs shall be expanded into spaces. If OFILL is set, two fill characters shall be transmitted for any delay.

Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be transmitted.

The actual delays depend on line speed and system load.

The initial output control value after *open()* is implementation-defined.

11.2.4 Control Modes

The *c_flag* field describes the hardware control of the terminal, and is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in **<termios.h>**; not all values specified are required to be supported by the underlying hardware:

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

In addition, the input and output baud rates are stored in the **termios** structure. The symbols in the following table are defined in **<termios.h>**. Not all values specified are required to be supported by the underlying hardware.

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

The following functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*. The effects on the terminal device shall not become effective and not all errors need be detected until the *tcsetattr()* function is successfully called.

The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-

6997 order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read.
 6998 CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used;
 6999 otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

7000 If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

7001 If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to
 7002 each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity
 7003 shall be used.

7004 If HUPCL is set, the modem control lines for the port shall be lowered when the last process
 7005 with the port open closes the port or the process terminates. The modem connection shall be
 7006 broken.

7007 If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If
 7008 CLOCAL is clear, the modem status lines shall be monitored.

7009 Under normal circumstances, a call to the *open()* function shall wait for the modem connection
 7010 to complete. However, if the O_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set,
 7011 the *open()* function shall return immediately without waiting for the connection.

7012 If the object for which the control modes are set is not an asynchronous serial connection, some
 7013 of the modes may be ignored; for example, if an attempt is made to set the baud rate on a
 7014 network connection to a terminal on another host, the baud rate need not be set on the
 7015 connection between that terminal and the machine to which it is directly connected.

7016 The initial hardware control value after *open()* is implementation-defined.

7017 11.2.5 Local Modes

7018 The *c_lflag* field of the argument structure is used to control various functions. It is composed of
 7019 the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
 7020 symbols in this table are defined in **<termios.h>**; not all values specified are required to be
 7021 supported by the underlying hardware:

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.

7033 If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input
 7034 characters shall not be echoed.

7035 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if
 7036 possible, the last character in the current line from the display. If there is no character to erase, an
 7037 implementation may echo an indication that this was the case, or do nothing.

7038 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the
 7039 line from the display or shall echo the newline character after the KILL character.

7040 If ECHONL and ICANON are set, the newline character shall be echoed even if ECHO is not set.

7041 If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit
7042 functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as
7043 described in Section 11.1.6 (on page 189).

7044 If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall
7045 not be satisfied until at least MIN bytes have been received or the timeout value TIME expired
7046 between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 190) for
7047 more details.

7048 If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is
7049 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF.
7050 If IEXTEN is not set, implementation-defined functions shall not be recognized and the
7051 corresponding input characters are processed as described for ICANON, ISIG, IXON, and
7052 IXOFF.

7053 If ISIG is set, each input character shall be checked against the special control characters INTR,
7054 QUIT, and SUSP. If an input character matches one of these control characters, the function
7055 associated with that character shall be performed. If ISIG is not set, no checking shall be done.
7056 Thus these special input functions are possible only if ISIG is set.

7057 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,
7058 QUIT, and SUSP characters shall not be done.

7059 If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to
7060 write to its controlling terminal if it is not in the foreground process group for that terminal. This
7061 signal, by default, stops the members of the process group. Otherwise, the output generated by
7062 that process shall be output to the current output stream. Processes that are blocking or ignoring
7063 SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal shall
7064 not be sent.

7065 The initial local control value after *open()* is implementation-defined.

7066 11.2.6 Special Control Characters

7067 The special control character values shall be defined by the array *c_cc*. The subscript name and
7068 description for each element in both canonical and non-canonical modes are as follows:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF	VINTR	EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR		INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

The subscript values are unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

Implementations that do not support changing the START and STOP characters may ignore the character values in the `c_cc` array indexed by the VSTART and VSTOP subscripts when `tcsetattr()` is called, but shall return the value in use when `tcgetattr()` is called.

The initial values of all control characters are implementation-defined.

If the value of one of the changeable special control characters (see Section 11.1.9 (on page 191)) is `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special meaning for the VMIN and VTIME entries of the `c_cc` array.

7094

Utility Conventions

7095

12.1 Utility Argument Syntax

This section describes the argument syntax of the standard utilities and introduces terminology used throughout IEEE Std 1003.1-2001 for describing the arguments processed by the utilities.

Within IEEE Std 1003.1-2001, a special notation is used for describing the syntax of a utility's arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated by this example (see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.9.1, Simple Commands):

```
utility_name [-a] [-b] [-c option_argument]
             [-d|-e] [-f option_argument] [operand...]
```

The notation used for the SYNOPSIS sections imposes requirements on the implementors of the standard utilities and provides a simple reference for the application developer or system user.

1. The utility in the example is named *utility_name*. It is followed by options, option-arguments, and operands. The arguments that consist of hyphens and single letters or digits, such as 'a', are known as "options" (or, historically, "flags"). Certain options are followed by an "option-argument", as shown with [-c option_argument]. The arguments following the last options and option-arguments are named "operands".
2. Option-arguments are sometimes shown separated from their options by <blank>s, sometimes directly adjacent. This reflects the situation that in some cases an option-argument is included within the same argument string as the option; in most cases it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page 203) require that the option be a separate argument from its option-argument, but there are some exceptions in IEEE Std 1003.1-2001 to ensure continued operation of historical applications:
 - a. If the SYNOPSIS of a standard utility shows a <space> between an option and option-argument (as with [-c option_argument] in the example), a conforming application shall use separate arguments for that option and its option-argument.
 - b. If a <space> is not shown (as with [-f option_argument] in the example), a conforming application shall place an option and its option-argument directly adjacent in the same argument string, without intervening <blank>s.
 - c. Notwithstanding the preceding requirements on conforming applications, a conforming implementation shall permit an application to specify options and option-arguments as a single argument or as separate arguments whether or not a <space> is shown on the synopsis line, except in those cases (marked with the XSI portability warning) where an option-argument is optional and no separation can be used.
 - d. A standard utility may also be implemented to operate correctly when the required separation into multiple arguments is violated by a non-conforming application.
3. Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of Section 12.2 (on page 203) applies. If an option that

does not have option-arguments is repeated, the results are undefined, unless otherwise stated.

4. Frequently, names of parameters that require substitution by actual values are shown with embedded underscores. Alternatively, parameters are shown as follows:

<parameter name>

The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and conforming applications shall not include them in data submitted to the utility.

5. When a utility has only a few permissible options, they are sometimes shown individually, as in the example. Utilities with many flags generally show all of the individual flags (that do not take option-arguments) grouped, as in:

utility_name [-abcDxyz] [-p arg] [operand]

Utilities with very complex arguments may be shown as follows:

utility_name [options] [operands]

6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:

- The number is interpreted as a decimal integer.
- Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.
- When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.
- Ranges greater than those listed here are allowed.

This does not mean that all numbers within the allowable range are necessarily semantically correct. A standard utility that accepts an option-argument or operand that is to be interpreted as a number, and for which a range of values smaller than that shown above is permitted by the IEEE Std 1003.1-2001, describes that smaller range along with the description of the option-argument or operand. If an error is generated, the utility's diagnostic message shall indicate that the value is out of the supported range, not that it is syntactically incorrect.

7. Arguments or option-arguments enclosed in the ' [' and '] ' notation are optional and can be omitted. Conforming applications shall not include the ' [' and '] ' symbols in data submitted to the utility.
8. Arguments separated by the ' | ' vertical bar notation are mutually-exclusive. Conforming applications shall not include the ' | ' symbol in data submitted to the utility. Alternatively, mutually-exclusive options and operands may be listed with multiple synopsis lines. For example:

*utility_name -d[-a] [-c option_argument] [operand...]
utility_name [-a] [-b] [operand...]*

When multiple synopsis lines are given for a utility, it is an indication that the utility has mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality of the utility so that only certain other arguments are valid in combination with one of the mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS section, the relationships between arguments depicted in the SYNOPSIS sections are

mandatory requirements placed on conforming applications. The use of conflicting mutually-exclusive arguments produces undefined results, unless a utility description specifies otherwise. When an option is shown without the ' [' and '] ' brackets, it means that option is required for that version of the SYNOPSIS. However, it is not required to be the first argument, as shown in the example above, unless otherwise stated.

9. Ellipses (" . . . ") are used to denote that one or more occurrences of an option or operand are allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero or more options or operands can be specified. The forms:

```
utility_name -f option_argument... [operand...]
utility_name [-g option_argument]... [operand...]
```

indicate that multiple occurrences of the option and its option-argument preceding the ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See also Guideline 11 in Section 12.2.) In the first example, each option-argument requires a preceding **-f** and at least one **-f option_argument** must be given.

10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities volume of IEEE Std 1003.1-2001, the indented lines following the initial line are continuation lines. An actual use of the command would appear on a single logical line.

12.2 Utility Syntax Guidelines

The following guidelines are established for the naming of utilities and for the specification of options, option-arguments, and operands. The *getopt()* function in the System Interfaces volume of IEEE Std 1003.1-2001 assists utilities in handling options and operands that conform to these guidelines.

Operands and option-arguments can contain characters not specified in the portable character set.

The guidelines are intended to provide guidance to the authors of future utilities, such as those written specific to a local system or that are components of a larger application. Some of the standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections describe the deviations.

Guideline 1: Utility names should be between two and nine characters, inclusive.

Guideline 2: Utility names should include lowercase letters (the **lower** character classification) and digits only from the portable character set.

Guideline 3: Each option name should be a single alphanumeric character (the **alnum** character classification) from the portable character set. The **-W** (capital-W) option shall be reserved for vendor options.

Multi-digit options should not be allowed.

Guideline 4: All options should be preceded by the **'-'** delimiter character.

Guideline 5: Options without option-arguments should be accepted when grouped behind one **'-'** delimiter.

Guideline 6: Each option and option-argument should be a separate argument, except as noted in Section 12.1 (on page 201), item (2).

Guideline 7: Option-arguments should not be optional.

7221	Guideline 8:	When multiple option-arguments are specified to follow a single option, they should be presented as a single argument, using commas within that argument or <blank>s within that argument to separate them.	
7222			
7223			
7224	Guideline 9:	All options should precede operands on the command line.	
7225	Guideline 10:	The argument -- should be accepted as a delimiter indicating the end of options. Any following arguments should be treated as operands, even if they begin with the '-' character. The -- argument should not be used as an option or as an operand.	
7226			
7227			
7228			
7229	Guideline 11:	The order of different options relative to one another should not matter, unless the options are documented as mutually-exclusive and such an option is documented to override any incompatible options preceding it. If an option that has option-arguments is repeated, the option and option-argument combinations should be interpreted in the order specified on the command line.	
7230			
7231			
7232			
7233			
7234			
7235	Guideline 12:	The order of operands may matter and position-related interpretations should be determined on a utility-specific basis.	
7236			
7237	Guideline 13:	For utilities that use operands to represent files to be opened for either reading or writing, the '-' operand should be used only to mean standard input (or standard output when it is clear from context that an output file is being specified).	
7238			
7239			
7240			
7241	The utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 that claim conformance to these guidelines shall conform completely to these guidelines as if these guidelines contained the term “shall” instead of “should”. On some implementations, the utilities accept usage in violation of these guidelines for backwards-compatibility as well as accepting the required form.		
7242			
7243			
7244			
7245	It is recommended that all future utilities and applications use these guidelines to enhance user portability. The fact that some historical utilities could not be changed (to avoid breaking existing applications) should not deter this future goal.		
7246			
7247			

Headers

7248

7249 This chapter describes the contents of headers.

7250 Headers contain function prototypes, the definition of symbolic constants, common structures,
 7251 preprocessor macros, and defined types. Each function in the System Interfaces volume of
 7252 IEEE Std 1003.1-2001 specifies the headers that an application shall include in order to use that
 7253 function. In most cases, only one header is required. These headers are present on an application
 7254 development system; they need not be present on the target execution system.

7255 13.1 Format of Entries

7256 The entries in this chapter are based on a common format as follows. The only sections relating
 7257 to conformance are the SYNOPSIS and DESCRIPTION.

7258 NAME

7259 This section gives the name or names of the entry and briefly states its purpose.

7260 SYNOPSIS

7261 This section summarizes the use of the entry being described.

7262 DESCRIPTION

7263 This section describes the functionality of the header.

7264 APPLICATION USAGE

7265 This section is informative.

7266 This section gives warnings and advice to application writers about the entry. In the
 7267 event of conflict between warnings and advice and a normative part of this volume of
 7268 IEEE Std 1003.1-2001, the normative material is to be taken as correct.

7269 RATIONALE

7270 This section is informative.

7271 This section contains historical information concerning the contents of this volume of
 7272 IEEE Std 1003.1-2001 and why features were included or discarded by the standard
 7273 developers.

7274 FUTURE DIRECTIONS

7275 This section is informative.

7276 This section provides comments which should be used as a guide to current thinking;
 7277 there is not necessarily a commitment to adopt these future directions.

7278 SEE ALSO

7279 This section is informative.

7280 This section gives references to related information.

7281 CHANGE HISTORY

7282 This section is informative.

7283 This section shows the derivation of the entry and any significant changes that have
 7284 been made to it.

7285 **NAME**7286 aio.h — asynchronous input and output (**REALTIME**)7287 **SYNOPSIS**

7288 AIO #include <aio.h>

7289

7290 **DESCRIPTION**7291 The <aio.h> header shall define the **aio** structure which shall include at least the following
7292 members:

7293	int	aio_fildes	File descriptor.
7294	off_t	aio_offset	File offset.
7295	volatile void	*aio_buf	Location of buffer.
7296	size_t	aio_nbytes	Length of transfer.
7297	int	aio_reqprio	Request priority offset.
7298	struct sigevent	aio_sigevent	Signal number and value.
7299	int	aio_lio_opcode	Operation to be performed.

7300 This header shall also include the following constants:

7301 AIO_ALLDONE A return value indicating that none of the requested operations could be
7302 canceled since they are already complete.7303 AIO_CANCELED A return value indicating that all requested operations have been
7304 canceled.

7305 AIO_NOTCANCELED

7306 A return value indicating that some of the requested operations could not
7307 be canceled since they are in progress.7308 LIO_NOP A *lio_listio()* element operation option indicating that no transfer is
7309 requested.7310 LIO_NOWAIT A *lio_listio()* synchronization operation indicating that the calling thread
7311 is to continue execution while the *lio_listio()* operation is being
7312 performed, and no notification is given when the operation is complete.7313 LIO_READ A *lio_listio()* element operation option requesting a read.7314 LIO_WAIT A *lio_listio()* synchronization operation indicating that the calling thread
7315 is to suspend until the *lio_listio()* operation is complete.7316 LIO_WRITE A *lio_listio()* element operation option requesting a write.7317 The following shall be declared as functions and may also be defined as macros. Function
7318 prototypes shall be provided.

```

7319 int      aio_cancel(int, struct aiocb *);
7320 int      aio_error(const struct aiocb *);
7321 int      aio_fsync(int, struct aiocb *);
7322 int      aio_read(struct aiocb *);
7323 ssize_t  aio_return(struct aiocb *);
7324 int      aio_suspend(const struct aiocb *const[], int,
7325                     const struct timespec *);
7326 int      aio_write(struct aiocb *);
7327 int      lio_listio(int, struct aiocb *restrict const[restrict], int,
7328                     struct sigevent *restrict);

```

7329 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>,
7330 <signal.h>, <sys/types.h>, and <time.h>.

7331 **APPLICATION USAGE**

7332 None.

7333 **RATIONALE**

7334 None.

7335 **FUTURE DIRECTIONS**

7336 None.

7337 **SEE ALSO**

7338 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
7339 IEEE Std 1003.1-2001, *fsync()*, *lseek()*, *read()*, *write()*

7340 **CHANGE HISTORY**

7341 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7342 **Issue 6**

7343 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7344 The description of the constants is expanded.

7345 The **restrict** keyword is added to the prototype for *lio_listio()*.

7346 **NAME**

7347 arpa/inet.h — definitions for internet operations

7348 **SYNOPSIS**

7349 #include <arpa/inet.h>

7350 **DESCRIPTION**7351 The **in_port_t** and **in_addr_t** types shall be defined as described in <netinet/in.h>.7352 The **in_addr** structure shall be defined as described in <netinet/in.h>.7353 IP6 The INET_ADDRSTRLEN and INET6_ADDRSTRLEN macros shall be defined as described in
7354 <netinet/in.h>.7355 The following shall either be declared as functions, defined as macros, or both. If functions are
7356 declared, function prototypes shall be provided.

7357 uint32_t htonl(uint32_t);

7358 uint16_t htons(uint16_t);

7359 uint32_t ntohl(uint32_t);

7360 uint16_t ntohs(uint16_t);

7361 The **uint32_t** and **uint16_t** types shall be defined as described in <inttypes.h>.7362 The following shall be declared as functions and may also be defined as macros. Function
7363 prototypes shall be provided.

7364 in_addr_t inet_addr(const char *);

7365 char *inet_ntoa(struct in_addr);

7366 const char *inet_ntop(int, const void *restrict, char *restrict,
7367 socklen_t);

7368 int inet_pton(int, const char *restrict, void *restrict);

7369 Inclusion of the <arpa/inet.h> header may also make visible all symbols from <netinet/in.h>
7370 and <inttypes.h>.7371 **APPLICATION USAGE**

7372 None.

7373 **RATIONALE**

7374 None.

7375 **FUTURE DIRECTIONS**

7376 None.

7377 **SEE ALSO**7378 <netinet/in.h>, <inttypes.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *htonl()*,
7379 *inet_addr()*7380 **CHANGE HISTORY**

7381 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7382 The **restrict** keyword is added to the prototypes for *inet_ntop()* and *inet_pton()*.

7383 **NAME**

7384 assert.h — verify program assertion

7385 **SYNOPSIS**

7386 #include <assert.h>

7387 **DESCRIPTION**

7388 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 7389 conflict between the requirements described here and the ISO C standard is unintentional. This
 7390 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7391 The <assert.h> header shall define the *assert()* macro. It refers to the macro NDEBUG which is
 7392 not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this
 7393 header, the *assert()* macro shall be defined simply as:

7394 #define assert(ignore) ((void) 0)

7395 Otherwise, the macro behaves as described in *assert()*.

7396 The *assert()* macro shall be redefined according to the current state of NDEBUG each time
 7397 <assert.h> is included.

7398 The *assert()* macro shall be implemented as a macro, not as a function. If the macro definition is
 7399 suppressed in order to access an actual function, the behavior is undefined.

7400 **APPLICATION USAGE**

7401 None.

7402 **RATIONALE**

7403 None.

7404 **FUTURE DIRECTIONS**

7405 None.

7406 **SEE ALSO**7407 The System Interfaces volume of IEEE Std 1003.1-2001, *assert()*7408 **CHANGE HISTORY**

7409 First released in Issue 1. Derived from Issue 1 of the SVID.

7410 **Issue 6**

7411 The definition of the *assert()* macro is changed for alignment with the ISO/IEC 9899:1999
 7412 standard.

7413 **NAME**

7414 complex.h — complex arithmetic

7415 **SYNOPSIS**

7416 #include <complex.h>

7417 **DESCRIPTION**

7418 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 7419 conflict between the requirements described here and the ISO C standard is unintentional. This
 7420 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7421 The <complex.h> header shall define the following macros:

7422 complex Expands to **_Complex**.

7423 **_Complex_I** Expands to a constant expression of type **const float _Complex**, with the
 7424 value of the imaginary unit (that is, a number i such that $i^2=-1$).

7425 imaginary Expands to **_Imaginary**.

7426 **_Imaginary_I** Expands to a constant expression of type **const float _Imaginary** with the
 7427 value of the imaginary unit.

7428 **I** Expands to either **_Imaginary_I** or **_Complex_I**. If **_Imaginary_I** is not defined,
 7429 **I** expands to **_Complex_I**.

7430 The macros **imaginary** and **_Imaginary_I** shall be defined if and only if the implementation
 7431 supports imaginary types.

7432 An application may undefine and then, perhaps, redefine the **complex**, **imaginary**, and **I** macros.

7433 The following shall be declared as functions and may also be defined as macros. Function
 7434 prototypes shall be provided.

7435	double	cabs(double complex);
7436	float	cabsf(float complex);
7437	long double	cabsl(long double complex);
7438	double complex	cacos(double complex);
7439	float complex	cacosf(float complex);
7440	double complex	cacosh(double complex);
7441	float complex	cacoshf(float complex);
7442	long double complex	cacoshl(long double complex);
7443	long double complex	cacosl(long double complex);
7444	double	carg(double complex);
7445	float	cargf(float complex);
7446	long double	cargl(long double complex);
7447	double complex	casin(double complex);
7448	float complex	casinf(float complex);
7449	double complex	casinh(double complex);
7450	float complex	casinhf(float complex);
7451	long double complex	casinhl(long double complex);
7452	long double complex	casinl(long double complex);
7453	double complex	catan(double complex);
7454	float complex	catanf(float complex);
7455	double complex	catanh(double complex);
7456	float complex	catanhf(float complex);
7457	long double complex	catanhl(long double complex);
7458	long double complex	catanl(long double complex);


```

7459     double complex      ccos(double complex);
7460     float complex       ccosf(float complex);
7461     double complex      ccosh(double complex);
7462     float complex       ccoshf(float complex);
7463     long double complex ccoshl(long double complex);
7464     long double complex ccosl(long double complex);
7465     double complex      cexp(double complex);
7466     float complex       cexpf(float complex);
7467     long double complex cexpl(long double complex);
7468     double              cimag(double complex);
7469     float              cimagf(float complex);
7470     long double        cimagl(long double complex);
7471     double complex      clog(double complex);
7472     float complex       clogf(float complex);
7473     long double complex clogl(long double complex);
7474     double complex      conj(double complex);
7475     float complex       conjf(float complex);
7476     long double complex conjl(long double complex);
7477     double complex      cpow(double complex, double complex);
7478     float complex       cpowf(float complex, float complex);
7479     long double complex cpowl(long double complex, long double complex);
7480     double complex      cproj(double complex);
7481     float complex       cprojf(float complex);
7482     long double complex cprojl(long double complex);
7483     double              creal(double complex);
7484     float              crealf(float complex);
7485     long double        creall(long double complex);
7486     double complex      csin(double complex);
7487     float complex       csinf(float complex);
7488     double complex      csinh(double complex);
7489     float complex       csinhf(float complex);
7490     long double complex csinhl(long double complex);
7491     long double complex csinl(long double complex);
7492     double complex      csqrt(double complex);
7493     float complex       csqrtf(float complex);
7494     long double complex csqrtl(long double complex);
7495     double complex      ctan(double complex);
7496     float complex       ctanf(float complex);
7497     double complex      ctanh(double complex);
7498     float complex       ctanhf(float complex);
7499     long double complex ctanhl(long double complex);
7500     long double complex ctanl(long double complex);

```

7501 APPLICATION USAGE

7502 Values are interpreted as radians, not degrees.

7503 RATIONALE

7504 The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the
7505 identifier *i* for other purposes. The application can use a different identifier, say *j*, for the
7506 imaginary unit by following the inclusion of the <complex.h> header with:

```

7507     #undef I
7508     #define j _Imaginary_I

```

7509 An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a
 7510 sufficiently convenient and more generally useful notation for imaginary terms. The
 7511 corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or
 7512 notational convenience will not result in widening types.

7513 On systems with imaginary types, the application has the ability to control whether use of the
 7514 macro `I` introduces an imaginary type, by explicitly defining `I` to be `_Imaginary_I` or `_Complex_I`.
 7515 Disallowing imaginary types is useful for some applications intended to run on implementations
 7516 without support for such types.

7517 The macro `_Imaginary_I` provides a test for whether imaginary types are supported.

7518 The `cis()` function ($\cos(x) + I\sin(x)$) was considered but rejected because its implementation is
 7519 easy and straightforward, even though some implementations could compute sine and cosine
 7520 more efficiently in tandem.

7521 FUTURE DIRECTIONS

7522 The following function names and the same names suffixed with *f* or *l* are reserved for future
 7523 use, and may be added to the declarations in the <complex.h> header.

7524	<code>cerf()</code>	<code>cexpm1()</code>	<code>clog2()</code>
7525	<code>cerfc()</code>	<code>clog10()</code>	<code>clgamma()</code>
7526	<code>cexp2()</code>	<code>clog1p()</code>	<code>ctgamma()</code>

7527 SEE ALSO

7528 The System Interfaces volume of IEEE Std 1003.1-2001, `cabs()`, `cacos()`, `cacosh()`, `carg()`, `casin()`,
 7529 `casinh()`, `catan()`, `catanh()`, `ccos()`, `ccosh()`, `cexp()`, `cimag()`, `clog()`, `conj()`, `cpow()`, `cproj()`, `creal()`,
 7530 `csin()`, `csinh()`, `csqrt()`, `ctan()`, `ctanh()`

7531 CHANGE HISTORY

7532 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7533 **NAME**

7534 cpio.h — cpio archive values

7535 **SYNOPSIS**

7536 xSI #include <cpio.h>

7537

7538 **DESCRIPTION**

7539 Values needed by the *c_mode* field of the *cpio* archive format are described as follows:

7540

	Name	Description	Value (Octal)
7541			
7542	C_IRUSR	Read by owner.	0000400
7543	C_IWUSR	Write by owner.	0000200
7544	C_IXUSR	Execute by owner.	0000100
7545	C_IRGRP	Read by group.	0000040
7546	C_IWGRP	Write by group.	0000020
7547	C_IXGRP	Execute by group.	0000010
7548	C_IROTH	Read by others.	0000004
7549	C_IWOTH	Write by others.	0000002
7550	C_IXOTH	Execute by others.	0000001
7551	C_ISUID	Set user ID.	0004000
7552	C_ISGID	Set group ID.	0002000
7553	C_ISVTX	On directories, restricted deletion flag.	0001000
7554	C_ISDIR	Directory.	0040000
7555	C_ISFIFO	FIFO.	0010000
7556	C_ISREG	Regular file.	0100000
7557	C_ISBLK	Block special.	0060000
7558	C_ISCHR	Character special.	0020000
7559	C_ISCTG	Reserved.	0110000
7560	C_ISLNK	Symbolic link.	0120000
7561	C_ISSOCK	Socket.	0140000

7562 The header shall define the symbolic constant:

7563 MAGIC "070707"

7564 **APPLICATION USAGE**

7565 None.

7566 **RATIONALE**

7567 None.

7568 **FUTURE DIRECTIONS**

7569 None.

7570 **SEE ALSO**

7571 The Shell and Utilities volume of IEEE Std 1003.1-2001, *pax*

7572 **CHANGE HISTORY**

7573 First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988
7574 standard.

7575 **Issue 6**

7576 The SEE ALSO is updated to refer to *pax*, since the *cpio* utility is not included in the Shell and
7577 Utilities volume of IEEE Std 1003.1-2001.

7578 **NAME**

7579 ctype.h — character types

7580 **SYNOPSIS**

7581 #include <ctype.h>

7582 **DESCRIPTION**

7583 CX Some of the functionality described on this reference page extends the ISO C standard.
 7584 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 7585 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 7586 symbols in this header.

7587 The following shall be declared as functions and may also be defined as macros. Function
 7588 prototypes shall be provided.

```

7589       int    isalnum(int);
7590       int    isalpha(int);
7591 XSI       int    isascii(int);
7592       int    isblank(int);
7593       int    iscntrl(int);
7594       int    isdigit(int);
7595       int    isgraph(int);
7596       int    islower(int);
7597       int    isprint(int);
7598       int    ispunct(int);
7599       int    isspace(int);
7600       int    isupper(int);
7601       int    isxdigit(int);
7602 XSI       int    toascii(int);
7603       int    tolower(int);
7604       int    toupper(int);

```

7605 The following are defined as macros:

```

7606 XSI       int    _toupper(int);
7607       int    _tolower(int);
7608

```

7609 **APPLICATION USAGE**

7610 None.

7611 **RATIONALE**

7612 None.

7613 **FUTURE DIRECTIONS**

7614 None.

7615 **SEE ALSO**

7616 <locale.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *isalnum()*, *isalpha()*, *isascii()*,
 7617 *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*,
 7618 *mbstowcs()*, *mbtowc()*, *setlocale()*, *toascii()*, *tolower()*, *_tolower()*, *toupper()*, *_toupper()*, *wcstombs()*,
 7619 *wctomb()*

7620 **CHANGE HISTORY**

7621 First released in Issue 1. Derived from Issue 1 of the SVID.

7622 **Issue 6**

7623 Extensions beyond the ISO C standard are marked.

7624 **NAME**

7625 dirent.h — format of directory entries

7626 **SYNOPSIS**

7627 #include <dirent.h>

7628 **DESCRIPTION**

7629 The internal format of directories is unspecified.

7630 The <**dirent.h**> header shall define the following type:7631 **DIR** A type representing a directory stream.7632 It shall also define the structure **dirent** which shall include the following members:

7633 XSI ino_t d_ino File serial number.

7634 char d_name[] Name of entry.

7635 XSI The type **ino_t** shall be defined as described in <**sys/types.h**>.7636 The character array *d_name* is of unspecified size, but the number of bytes preceding the
7637 terminating null byte shall not exceed {NAME_MAX}.7638 The following shall be declared as functions and may also be defined as macros. Function
7639 prototypes shall be provided.

7640 int closedir(DIR *);

7641 DIR *opendir(const char *);

7642 struct dirent *readdir(DIR *);

7643 TSF int readdir_r(DIR *restrict, struct dirent *restrict,
7644 struct dirent **restrict);

7645 void rewinddir(DIR *);

7646 XSI void seekdir(DIR *, long);

7647 long telldir(DIR *);

7648

7649 **APPLICATION USAGE**

7650 None.

7651 **RATIONALE**7652 Information similar to that in the <**dirent.h**> header is contained in a file <**sys/dir.h**> in 4.2 BSD
7653 and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of
7654 IEEE Std 1003.1-2001 is **struct direct**. The filename was changed because the name <**sys/dir.h**>
7655 was also used in earlier implementations to refer to definitions related to the older access
7656 method; this produced name conflicts. The name of the structure was changed because this
7657 volume of IEEE Std 1003.1-2001 does not completely define what is in the structure, so it could
7658 be different on some implementations from **struct direct**.7659 The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:

7660 sizeof(d_name)

7661 is incorrect; use:

7662 strlen(d_name)

7663 instead.

7664 The array of **char** *d_name* is not a fixed size. Implementations may need to declare **struct dirent**
7665 with an array size for *d_name* of 1, but the actual number of characters provided matches (or
7666 only slightly exceeds) the length of the filename.

7667 **FUTURE DIRECTIONS**

7668 None.

7669 **SEE ALSO**

7670 <**sys/types.h**>, the System Interfaces volume of IEEE Std 1003.1-2001, *closedir()*, *opendir()*,
7671 *readdir()*, *readdir_r()*, *rewinddir()*, *seekdir()*, *telldir()*

7672 **CHANGE HISTORY**

7673 First released in Issue 2.

7674 **Issue 5**

7675 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

7676 **Issue 6**7677 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r()*.7678 The **restrict** keyword is added to the prototype for *readdir_r()*.

7679 **NAME**

7680 dlfcn.h — dynamic linking

7681 **SYNOPSIS**

7682 xSI #include <dlfcn.h>

7683

7684 **DESCRIPTION**7685 The <dlfcn.h> header shall define at least the following macros for use in the construction of a
7686 *dlopen()* mode argument:

7687 RTLD_LAZY Relocations are performed at an implementation-defined time.

7688 RTLD_NOW Relocations are performed when the object is loaded.

7689 RTLD_GLOBAL All symbols are available for relocation processing of other modules.

7690 RTLD_LOCAL All symbols are not made available for relocation processing by other
7691 modules.7692 The following shall be declared as functions and may also be defined as macros. Function
7693 prototypes shall be provided.

7694 int dlclose(void *);

7695 char *dlerror(void);

7696 void *dlopen(const char *, int);

7697 void *dlsym(void *restrict, const char *restrict);

7698 **APPLICATION USAGE**

7699 None.

7700 **RATIONALE**

7701 None.

7702 **FUTURE DIRECTIONS**

7703 None.

7704 **SEE ALSO**7705 The System Interfaces volume of IEEE Std 1003.1-2001, *dlopen()*, *dlclose()*, *dlsym()*, *dlerror()*7706 **CHANGE HISTORY**

7707 First released in Issue 5.

7708 **Issue 6**7709 The **restrict** keyword is added to the prototype for *dlsym()*.

7710 **NAME**

7711 errno.h — system error numbers

7712 **SYNOPSIS**

7713 #include <errno.h>

7714 **DESCRIPTION**

7715 CX Some of the functionality described on this reference page extends the ISO C standard. Any
7716 conflict between the requirements described here and the ISO C standard is unintentional. This
7717 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7718 CX The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

7719 The <errno.h> header shall provide a declaration for *errno* and give positive values for the
7720 following symbolic constants. Their values shall be unique except as noted below.

7721 [E2BIG] Argument list too long.

7722 [EACCES] Permission denied.

7723 [EADDRINUSE] Address in use.

7724 [EADDRNOTAVAIL] Address not available.

7725 [EAFNOSUPPORT] Address family not supported.

7726 [EAGAIN] Resource unavailable, try again (may be the same value as
7727 [EWOULDBLOCK]).

7728 [EALREADY] Connection already in progress.

7729 [EBADF] Bad file descriptor.

7730 [EBADMSG] Bad message.

7731 [EBUSY] Device or resource busy.

7732 [ECANCELED] Operation canceled.

7733 [ECHILD] No child processes.

7734 [ECONNABORTED] Connection aborted.

7735 [ECONNREFUSED] Connection refused.

7736 [ECONNRESET] Connection reset.

7737 [EDEADLK] Resource deadlock would occur.

7738 [EDESTADDRREQ] Destination address required.

7739 [EDOM] Mathematics argument out of domain of function.

7740 [EDQUOT] Reserved.

7741 [EEXIST] File exists.

7742 [EFAULT] Bad address.

7743 [EFBIG] File too large.

7744 [EHOSTUNREACH] Host is unreachable.

7745 [EIDRM] Identifier removed.

7746 [EILSEQ] Illegal byte sequence.

7747	[EINPROGRESS]	Operation in progress.
7748	[EINTR]	Interrupted function.
7749	[EINVAL]	Invalid argument.
7750	[EIO]	I/O error.
7751	[EISCONN]	Socket is connected.
7752	[EISDIR]	Is a directory.
7753	[ELOOP]	Too many levels of symbolic links.
7754	[EMFILE]	Too many open files.
7755	[EMLINK]	Too many links.
7756	[EMSGSIZE]	Message too large.
7757	[EMULTIHOP]	Reserved.
7758	[ENAMETOOLONG]	Filename too long.
7759	[ENETDOWN]	Network is down.
7760	[ENETRESET]	Connection aborted by network.
7761	[ENETUNREACH]	Network unreachable.
7762	[ENFILE]	Too many files open in system.
7763	[ENOBUFS]	No buffer space available.
7764	XSR [ENODATA]	No message is available on the STREAM head read queue.
7765	[ENODEV]	No such device.
7766	[ENOENT]	No such file or directory.
7767	[ENOEXEC]	Executable file format error.
7768	[ENOLCK]	No locks available.
7769	[ENOLINK]	Reserved.
7770	[ENOMEM]	Not enough space.
7771	[ENOMSG]	No message of the desired type.
7772	[ENOPROTOPT]	Protocol not available.
7773	[ENOSPC]	No space left on device.
7774	XSR [ENOSR]	No STREAM resources.
7775	XSR [ENOSTR]	Not a STREAM.
7776	[ENOSYS]	Function not supported.
7777	[ENOTCONN]	The socket is not connected.
7778	[ENOTDIR]	Not a directory.
7779	[ENOTEMPTY]	Directory not empty.
7780	[ENOTSOCK]	Not a socket.

7781	[ENOTSUP]	Not supported.
7782	[ENOTTY]	Inappropriate I/O control operation.
7783	[ENXIO]	No such device or address.
7784	[EOPNOTSUPP]	Operation not supported on socket.
7785	[EOVERFLOW]	Value too large to be stored in data type.
7786	[EPERM]	Operation not permitted.
7787	[EPIPE]	Broken pipe.
7788	[EPROTO]	Protocol error.
7789	[EPROTONOSUPPORT]	
7790		Protocol not supported.
7791	[EPROTOTYPE]	Protocol wrong type for socket.
7792	[ERANGE]	Result too large.
7793	[EROFS]	Read-only file system.
7794	[ESPIPE]	Invalid seek.
7795	[ESRCH]	No such process.
7796	[ESTALE]	Reserved.
7797	XSR [ETIME]	Stream <i>ioctl()</i> timeout.
7798	[ETIMEDOUT]	Connection timed out.
7799	[ETXTBSY]	Text file busy.
7800	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
7801	[EXDEV]	Cross-device link.

7802 APPLICATION USAGE

7803 Additional error numbers may be defined on conforming systems; see the System Interfaces
7804 volume of IEEE Std 1003.1-2001.

7805 RATIONALE

7806 None.

7807 FUTURE DIRECTIONS

7808 None.

7809 SEE ALSO

7810 The System Interfaces volume of IEEE Std 1003.1-2001, Section 2.3, Error Numbers

7811 CHANGE HISTORY

7812 First released in Issue 1. Derived from Issue 1 of the SVID.

7813 Issue 5

7814 Updated for alignment with the POSIX Realtime Extension.

7815 Issue 6

7816 The following new requirements on POSIX implementations derive from alignment with the
7817 Single UNIX Specification:

- 7818 • The majority of the error conditions previously marked as extensions are now mandatory,
7819 except for the STREAMS-related error conditions.

7820 Values for *errno* are now required to be distinct positive values rather than non-zero values. This
7821 change is for alignment with the ISO/IEC 9899:1999 standard.

7822 **NAME**

7823 fcntl.h — file control options

7824 **SYNOPSIS**

7825 #include <fcntl.h>

7826 **DESCRIPTION**

7827 The <fcntl.h> header shall define the following requests and arguments for use by the functions
7828 *fcntl()* and *open()*.

7829 Values for *cmd* used by *fcntl()* (the following values are unique) are as follows:

- 7830 F_DUPFD Duplicate file descriptor.
- 7831 F_GETFD Get file descriptor flags.
- 7832 F_SETFD Set file descriptor flags.
- 7833 F_GETFL Get file status flags and file access modes.
- 7834 F_SETFL Set file status flags.
- 7835 F_GETLK Get record locking information.
- 7836 F_SETLK Set record locking information.
- 7837 F_SETLKW Set record locking information; wait if blocked.
- 7838 F_GETOWN Get process or process group ID to receive SIGURG signals.
- 7839 F_SETOWN Set process or process group ID to receive SIGURG signals.

7840 File descriptor flags used for *fcntl()* are as follows:

- 7841 FD_CLOEXEC Close the file descriptor upon execution of an *exec* family function.

7842 Values for *l_type* used for record locking with *fcntl()* (the following values are unique) are as
7843 follows:

- 7844 F_RDLCK Shared or read lock.
- 7845 F_UNLCK Unlock.
- 7846 F_WRLCK Exclusive or write lock.

7847 XSI The values used for *l_whence*, *SEEK_SET*, *SEEK_CUR*, and *SEEK_END* shall be defined as
7848 described in <unistd.h>.

7849 The following values are file creation flags and are used in the *oflag* value to *open()*. They shall
7850 be bitwise-distinct.

- 7851 O_CREAT Create file if it does not exist.
- 7852 O_EXCL Exclusive use flag.
- 7853 O_NOCTTY Do not assign controlling terminal.
- 7854 O_TRUNC Truncate flag.

7855 File status flags used for *open()* and *fcntl()* are as follows:

- 7856 O_APPEND Set append mode.
- 7857 SIO O_DSYNC Write according to synchronized I/O data integrity completion.
- 7858 O_NONBLOCK Non-blocking mode.

7859 SIO **O_RSYNC** Synchronized read I/O operations.

7860 **O_SYNC** Write according to synchronized I/O file integrity completion.

7861 Mask for use with file access modes is as follows:

7862 **O_ACCMODE** Mask for file access modes.

7863 File access modes used for *open()* and *fcntl()* are as follows:

7864 **O_RDONLY** Open for reading only.

7865 **O_RDWR** Open for reading and writing.

7866 **O_WRONLY** Open for writing only.

7867 XSI The symbolic names for file modes for use as values of **mode_t** shall be defined as described in
7868 <sys/stat.h>.

7869 ADV Values for *advice* used by *posix_fadvise()* are as follows:

7870 **POSIX_FADV_NORMAL**

7871 The application has no advice to give on its behavior with respect to the specified data. It is
7872 the default characteristic if no advice is given for an open file.

7873 **POSIX_FADV_SEQUENTIAL**

7874 The application expects to access the specified data sequentially from lower offsets to
7875 higher offsets.

7876 **POSIX_FADV_RANDOM**

7877 The application expects to access the specified data in a random order.

7878 **POSIX_FADV_WILLNEED**

7879 The application expects to access the specified data in the near future.

7880 **POSIX_FADV_DONTNEED**

7881 The application expects that it will not access the specified data in the near future.

7882 **POSIX_FADV_NOREUSE**

7883 The application expects to access the specified data once and then not reuse it thereafter.
7884

7885 The structure **lock** describes a file lock. It shall include the following members:

7886 **short l_type** Type of lock; **F_RDLCK**, **F_WRLCK**, **F_UNLCK**.

7887 **short l_whence** Flag for starting offset.

7888 **off_t l_start** Relative offset in bytes.

7889 **off_t l_len** Size; if 0 then until EOF.

7890 **pid_t l_pid** Process ID of the process holding the lock; returned with **F_GETLK**.

7891 The **mode_t**, **off_t**, and **pid_t** types shall be defined as described in <sys/types.h>.

7892 The following shall be declared as functions and may also be defined as macros. Function
7893 prototypes shall be provided.

7894 **int creat(const char *, mode_t);**

7895 **int fcntl(int, int, ...);**

7896 **int open(const char *, int, ...);**

7897 ADV **int posix_fadvise(int, off_t, size_t, int);**

7898 **int posix_fallocate(int, off_t, size_t);**
7899

7900 XSI Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and
 7901 <unistd.h>.

7902 APPLICATION USAGE

7903 None.

7904 RATIONALE

7905 None.

7906 FUTURE DIRECTIONS

7907 None.

7908 SEE ALSO

7909 <sys/stat.h>, <sys/types.h>, <unistd.h>, the System Interfaces volume of IEEE Std 1003.1-2001,
 7910 *creat()*, *exec*, *fcntl()*, *open()*, *posix_fadvise()*, *posix_fallocate()*, *posix_madvise()*

7911 CHANGE HISTORY

7912 First released in Issue 1. Derived from Issue 1 of the SVID.

7913 Issue 5

7914 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

7915 Issue 6

7916 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 7917 • O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output option.

7918 The following new requirements on POSIX implementations derive from alignment with the
 7919 Single UNIX Specification:

- 7920 • The definition of the **mode_t**, **off_t**, and **pid_t** types is mandated.

7921 The F_GETOWN and F_SETOWN values are added for sockets.

7922 The *posix_fadvise()*, *posix_fallocate()*, and *posix_madvise()* functions are added for alignment with
 7923 IEEE Std 1003.1d-1999.

7924 IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for *posix_madvise()* to
 7925 <sys/mman.h>.

7926 **NAME**

7927 fenv.h — floating-point environment

7928 **SYNOPSIS**

7929 #include <fenv.h>

7930 **DESCRIPTION**

7931 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 7932 conflict between the requirements described here and the ISO C standard is unintentional. This
 7933 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7934 The <fenv.h> header shall define the following data types through **typedef**:

7935 **fenv_t** Represents the entire floating-point environment. The floating-point environment
 7936 refers collectively to any floating-point status flags and control modes supported
 7937 by the implementation.

7938 **fexcept_t** Represents the floating-point status flags collectively, including any status the
 7939 implementation associates with the flags. A floating-point status flag is a system
 7940 variable whose value is set (but never cleared) when a floating-point exception is
 7941 raised, which occurs as a side effect of exceptional floating-point arithmetic to
 7942 provide auxiliary information. A floating-point control mode is a system variable
 7943 whose value may be set by the user to affect the subsequent behavior of floating-
 7944 point arithmetic.

7945 The <fenv.h> header shall define the following constants if and only if the implementation
 7946 supports the floating-point exception by means of the floating-point functions *feclearexcept()*,
 7947 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. Each expands to an integer
 7948 constant expression with values such that bitwise-inclusive ORs of all combinations of the
 7949 constants result in distinct values.

7950 FE_DIVBYZERO
 7951 FE_INEXACT
 7952 FE_INVALID
 7953 FE_OVERFLOW
 7954 FE_UNDERFLOW

7955 The <fenv.h> header shall define the following constant, which is simply the bitwise-inclusive
 7956 OR of all floating-point exception constants defined above:

7957 FE_ALL_EXCEPT

7958 The <fenv.h> header shall define the following constants if and only if the implementation
 7959 supports getting and setting the represented rounding direction by means of the *fegetround()*
 7960 and *fesetround()* functions. Each expands to an integer constant expression whose values are
 7961 distinct non-negative values.

7962 FE_DOWNWARD
 7963 FE_TONEAREST
 7964 FE_TOWARDZERO
 7965 FE_UPWARD

7966 The <fenv.h> header shall define the following constant, which represents the default floating-
 7967 point environment (that is, the one installed at program startup) and has type pointer to const-
 7968 qualified **fenv_t**. It can be used as an argument to the functions within the <fenv.h> header that
 7969 manage the floating-point environment.

7970 FE_DFL_ENV

7971 The following shall be declared as functions and may also be defined as macros. Function
7972 prototypes shall be provided.

```
7973 int feclearexcept(int);
7974 int fegetexceptflag(fexcept_t *, int);
7975 int feraiseexcept(int);
7976 int fesetexceptflag(const fexcept_t *, int);
7977 int fetestexcept(int);
7978 int fegetround(void);
7979 int fesetround(int);
7980 int fegetenv(fenv_t *);
7981 int feholdexcept(fenv_t *);
7982 int fesetenv(const fenv_t *);
7983 int feupdateenv(const fenv_t *);
```

7984 The FENV_ACCESS pragma provides a means to inform the implementation when an
7985 application might access the floating-point environment to test floating-point status flags or run
7986 under non-default floating-point control modes. The pragma shall occur either outside external
7987 declarations or preceding all explicit declarations and statements inside a compound statement.
7988 When outside external declarations, the pragma takes effect from its occurrence until another
7989 FENV_ACCESS pragma is encountered, or until the end of the translation unit. When inside a
7990 compound statement, the pragma takes effect from its occurrence until another FENV_ACCESS
7991 pragma is encountered (including within a nested compound statement), or until the end of the
7992 compound statement; at the end of a compound statement the state for the pragma is restored to
7993 its condition just before the compound statement. If this pragma is used in any other context, the
7994 behavior is undefined. If part of an application tests floating-point status flags, sets floating-
7995 point control modes, or runs under non-default mode settings, but was translated with the state
7996 for the FENV_ACCESS pragma off, the behavior is undefined. The default state (on or off) for
7997 the pragma is implementation-defined. (When execution passes from a part of the application
7998 translated with FENV_ACCESS off to a part translated with FENV_ACCESS on, the state of the
7999 floating-point status flags is unspecified and the floating-point control modes have their default
8000 settings.)

8001 APPLICATION USAGE

8002 This header is designed to support the floating-point exception status flags and directed-
8003 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-
8004 point state information. Also it is designed to facilitate code portability among all systems.

8005 Certain application programming conventions support the intended model of use for the
8006 floating-point environment:

- 8007 • A function call does not alter its caller's floating-point control modes, clear its caller's
8008 floating-point status flags, nor depend on the state of its caller's floating-point status flags
8009 unless the function is so documented.
- 8010 • A function call is assumed to require default floating-point control modes, unless its
8011 documentation promises otherwise.
- 8012 • A function call is assumed to have the potential for raising floating-point exceptions, unless
8013 its documentation promises otherwise.

8014 With these conventions, an application can safely assume default floating-point control modes
8015 (or be unaware of them). The responsibilities associated with accessing the floating-point
8016 environment fall on the application that does so explicitly.

8017 Even though the rounding direction macros may expand to constants corresponding to the
8018 values of FLT_ROUNDS, they are not required to do so.

```

8019      For example:
8020      #include <fenv.h>
8021      void f(double x)
8022      {
8023          #pragma STDC FENV_ACCESS ON
8024          void g(double);
8025          void h(double);
8026          /* ... */
8027          g(x + 1);
8028          h(x + 1);
8029          /* ... */
8030      }

```

8031 If the function *g()* might depend on status flags set as a side effect of the first *x+1*, or if the
 8032 second *x+1* might depend on control modes set as a side effect of the call to function *g()*, then
 8033 the application shall contain an appropriately placed invocation as follows:

```

8034      #pragma STDC FENV_ACCESS ON

```

8035 RATIONALE

8036 The **fexcept_t** Type

8037 **fexcept_t** does not have to be an integer type. Its values must be obtained by a call to
 8038 *fegetexceptflag()*, and cannot be created by logical operations from the exception macros. An
 8039 implementation might simply implement **fexcept_t** as an **int** and use the representations
 8040 reflected by the exception macros, but is not required to; other representations might contain
 8041 extra information about the exceptions. **fexcept_t** might be a **struct** with a member for each
 8042 exception (that might hold the address of the first or last floating-point instruction that caused
 8043 that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an
 8044 **fexcept_t**, and so the user cannot inspect it.

8045 Exception and Rounding Macros

8046 Macros corresponding to unsupported modes and rounding directions are not defined by the
 8047 implementation and must not be defined by the application. An application might use **#ifdef** to
 8048 test for this.

8049 FUTURE DIRECTIONS

8050 None.

8051 SEE ALSO

8052 The System Interfaces volume of IEEE Std 1003.1-2001, *feclearexcept()*, *fegetenv()*, *fegetexceptflag()*,
 8053 *fegetround()*, *fehldexcept()*, *feraiseexcept()*, *fesetenv()*, *fesetexceptflag()*, *fesetround()*, *fetestexcept()*,
 8054 *feupdateenv()*

8055 CHANGE HISTORY

8056 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8057 The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*,
 8058 *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the
 8059 ISO/IEC 9899:1999 standard, Defect Report 202.

8060 **NAME**

8061 float.h — floating types

8062 **SYNOPSIS**

8063 #include <float.h>

8064 **DESCRIPTION**

8065 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
8066 conflict between the requirements described here and the ISO C standard is unintentional. This
8067 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8068 The characteristics of floating types are defined in terms of a model that describes a
8069 representation of floating-point numbers and values that provide information about an
8070 implementation's floating-point arithmetic.

8071 The following parameters are used to define the model for each floating-point type:

8072 *s* Sign (± 1).

8073 *b* Base or radix of exponent representation (an integer > 1).

8074 *e* Exponent (an integer between a minimum $e_{\min}$ and a maximum $e_{\max}$).

8075 *p* Precision (the number of base-*b* digits in the significand).

8076 *f_k* Non-negative integers less than *b* (the significand digits).

8077 A floating-point number *x* is defined by the following model:

$$8078 \quad x = sb^e \sum_{k=1}^p f_k b^{-k}, \quad e_{\min} \leq e \leq e_{\max}$$

8079 In addition to normalized floating-point numbers ($f_1 > 0$ if $x \neq 0$), floating types may be able to
8080 contain other kinds of floating-point numbers, such as subnormal floating-point numbers ($x \neq 0$,
8081 $e = e_{\min}$, $f_1 = 0$) and unnormalized floating-point numbers ($x \neq 0$, $e > e_{\min}$, $f_1 = 0$), and values that are
8082 not floating-point numbers, such as infinities and NaNs. A NaN is an encoding signifying Not-
8083 a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a
8084 floating-point exception; a *signaling NaN* generally raises a floating-point exception when
8085 occurring as an arithmetic operand.

8086 The accuracy of the floating-point operations ($' + '$, $' - '$, $' * '$, $' / '$) and of the library functions
8087 in <math.h> and <complex.h> that return floating-point results is implementation-defined. The
8088 implementation may state that the accuracy is unknown.

8089 All integer values in the <float.h> header, except FLT_ROUNDS, shall be constant expressions
8090 suitable for use in #if preprocessing directives; all floating values shall be constant expressions.
8091 All except DECIMAL_DIG, FLT_EVAL_METHOD, FLT_RADIX, and FLT_ROUNDS have
8092 separate names for all three floating-point types. The floating-point model representation is
8093 provided for all values except FLT_EVAL_METHOD and FLT_ROUNDS.

8094 The rounding mode for floating-point addition is characterized by the implementation-defined
8095 value of FLT_ROUNDS:

8096 -1 Indeterminable.

8097 0 Toward zero.

8098 1 To nearest.

8099 2 Toward positive infinity.

8100 3 Toward negative infinity.

8101 All other values for FLT_ROUND characterizes implementation-defined rounding behavior.

8102 The values of operations with floating operands and values subject to the usual arithmetic
8103 conversions and of floating constants are evaluated to a format whose range and precision may
8104 be greater than required by the type. The use of evaluation formats is characterized by the
8105 implementation-defined value of FLT_EVAL_METHOD:

8106 -1 Indeterminable.

8107 0 Evaluate all operations and constants just to the range and precision of the type.

8108 1 Evaluate operations and constants of type **float** and **double** to the range and precision of the
8109 **double** type; evaluate **long double** operations and constants to the range and precision of
8110 the **long double** type.

8111 2 Evaluate all operations and constants to the range and precision of the **long double** type.

8112 All other negative values for FLT_EVAL_METHOD characterize implementation-defined
8113 behavior.

8114 The values given in the following list shall be defined as constant expressions with
8115 implementation-defined values that are greater or equal in magnitude (absolute value) to those
8116 shown, with the same sign.

8117 • Radix of exponent representation, b .

8118 FLT_RADIX 2

8119 • Number of base-FLT_RADIX digits in the floating-point significand, p .

8120 FLT_MANT_DIG

8121 DBL_MANT_DIG

8122 LDBL_MANT_DIG

8123 • Number of decimal digits, n , such that any floating-point number in the widest supported
8124 floating type with $p_{\max}$ radix b digits can be rounded to a floating-point number with n
8125 decimal digits and back again without change to the value.

8126
$$\begin{cases} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil 1 + p_{\max} \log_{10} b \right\rceil & \text{otherwise} \end{cases}$$

8127 DECIMAL_DIG 10

8128 • Number of decimal digits, q , such that any floating-point number with q decimal digits can
8129 be rounded into a floating-point number with p radix b digits and back again without change
8130 to the q decimal digits.

8131
$$\begin{cases} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil (p - 1) \log_{10} b \right\rceil & \text{otherwise} \end{cases}$$

8132 FLT_DIG 6

8133 DBL_DIG 10

8134	LDBL_DIG	10
8135	• Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a	
8136	normalized floating-point number, $e_{\min}$.	
8137	FLT_MIN_EXP	
8138	DBL_MIN_EXP	
8139	LDBL_MIN_EXP	
8140	• Minimum negative integer such that 10 raised to that power is in the range of normalized	
8141	floating-point numbers.	
8142	$\left\lceil \log_{10} b^{e_{\min} - 1} \right\rceil$	
8143	FLT_MIN_10_EXP	−37
8144	DBL_MIN_10_EXP	−37
8145	LDBL_MIN_10_EXP	−37
8146	• Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable	
8147	finite floating-point number, $e_{\max}$.	
8148	FLT_MAX_EXP	
8149	DBL_MAX_EXP	
8150	LDBL_MAX_EXP	
8151	• Maximum integer such that 10 raised to that power is in the range of representable finite	
8152	floating-point numbers.	
8153	$\left\lceil \log_{10}((1 - b^{-p}) b^{e_{\max}}) \right\rceil$	
8154	FLT_MAX_10_EXP	+37
8155	DBL_MAX_10_EXP	+37
8156	LDBL_MAX_10_EXP	+37
8157	The values given in the following list shall be defined as constant expressions with	
8158	implementation-defined values that are greater than or equal to those shown:	
8159	• Maximum representable finite floating-point number.	
8160	$(1 - b^{-p}) b^{e_{\max}}$	
8161	FLT_MAX	1E+37
8162	DBL_MAX	1E+37
8163	LDBL_MAX	1E+37
8164	The values given in the following list shall be defined as constant expressions with	
8165	implementation-defined (positive) values that are less than or equal to those shown:	
8166	• The difference between 1 and the least value greater than 1 that is representable in the given	
8167	floating-point type, b^{1-p} .	
8168	FLT_EPSILON	1E−5
8169	DBL_EPSILON	1E−9

8170 LDBL_EPSILON 1E-9

8171 • Minimum normalized positive floating-point number, $b^{e_{\min}-1}$.

8172 FLT_MIN 1E-37

8173 DBL_MIN 1E-37

8174 LDBL_MIN 1E-37

8175 **APPLICATION USAGE**

8176 None.

8177 **RATIONALE**

8178 None.

8179 **FUTURE DIRECTIONS**

8180 None.

8181 **SEE ALSO**

8182 <complex.h>, <math.h>

8183 **CHANGE HISTORY**

8184 First released in Issue 4. Derived from the ISO C standard.

8185 **Issue 6**

8186 The description of the operations with floating-point values is updated for alignment with the
8187 ISO / IEC 9899:1999 standard.

8188 **NAME**

8189 fmtmsg.h — message display structures

8190 **SYNOPSIS**

8191 xSI #include <fmtmsg.h>

8192

8193 **DESCRIPTION**8194 The <fmtmsg.h> header shall define the following macros, which expand to constant integer
8195 expressions:

8196	MM_HARD	Source of the condition is hardware.
8197	MM_SOFT	Source of the condition is software.
8198	MM_FIRM	Source of the condition is firmware.
8199	MM_APPL	Condition detected by application.
8200	MM_UTIL	Condition detected by utility.
8201	MM_OPSYS	Condition detected by operating system.
8202	MM_RECOVER	Recoverable error.
8203	MM_NRECOV	Non-recoverable error.
8204	MM_HALT	Error causing application to halt.
8205	MM_ERROR	Application has encountered a non-fatal fault.
8206	MM_WARNING	Application has detected unusual non-error condition.
8207	MM_INFO	Informative message.
8208	MM_NOSEV	No severity level provided for the message.
8209	MM_PRINT	Display message on standard error.
8210	MM_CONSOLE	Display message on system console.

8211 The table below indicates the null values and identifiers for *fmtmsg()* arguments. The
8212 <fmtmsg.h> header shall define the macros in the **Identifier** column, which expand to constant
8213 expressions that expand to expressions of the type indicated in the **Type** column:

8214

8215

8216

8217

8218

8219

8220

8221

Argument	Type	Null-Value	Identifier
<i>label</i>	char *	(char*)0	MM_NULLLBL
<i>severity</i>	int	0	MM_NULLSEV
<i>class</i>	long	0L	MM_NULLMC
<i>text</i>	char *	(char*)0	MM_NULLTXT
<i>action</i>	char *	(char*)0	MM_NULLACT
<i>tag</i>	char *	(char*)0	MM_NULLTAG

8222 The <fmtmsg.h> header shall also define the following macros for use as return values for
8223 *fmtmsg()*:

8224	MM_OK	The function succeeded.
8225	MM_NOTOK	The function failed completely.
8226	MM_NOMSG	The function was unable to generate a message on standard error, but 8227 otherwise succeeded.

8228 MM_NOCON The function was unable to generate a console message, but otherwise
8229 succeeded.

8230 The following shall be declared as a function and may also be defined as a macro. A function
8231 prototype shall be provided.

8232 int fmtmsg(long, const char *, int,
8233 const char *, const char *, const char *);

8234 **APPLICATION USAGE**

8235 None.

8236 **RATIONALE**

8237 None.

8238 **FUTURE DIRECTIONS**

8239 None.

8240 **SEE ALSO**

8241 The System Interfaces volume of IEEE Std 1003.1-2001, *fmtmsg()*

8242 **CHANGE HISTORY**

8243 First released in Issue 4, Version 2.

8244 **NAME**

8245 fnmatch.h — filename-matching types

8246 **SYNOPSIS**

8247 #include <fnmatch.h>

8248 **DESCRIPTION**

8249 The <fnmatch.h> header shall define the following constants:

8250 FNM_NOMATCH The string does not match the specified pattern.

8251 FNM_PATHNAME Slash in *string* only matches slash in *pattern*.

8252 FNM_PERIOD Leading period in *string* must be exactly matched by period in *pattern*.

8253 FNM_NOESCAPE Disable backslash escaping.

8254 OB XSI FNM_NOSYS Reserved.

8255 The following shall be declared as a function and may also be defined as a macro. A function
8256 prototype shall be provided.

8257 int fnmatch(const char *, const char *, int);

8258 **APPLICATION USAGE**

8259 None.

8260 **RATIONALE**

8261 None.

8262 **FUTURE DIRECTIONS**

8263 None.

8264 **SEE ALSO**

8265 The System Interfaces volume of IEEE Std 1003.1-2001, *fnmatch()*, the Shell and Utilities volume
8266 of IEEE Std 1003.1-2001

8267 **CHANGE HISTORY**

8268 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8269 **Issue 6**

8270 The constant FNM_NOSYS is marked obsolescent.

8271 **NAME**

8272 ftw.h — file tree traversal

8273 **SYNOPSIS**

8274 xSI #include <ftw.h>

8275

8276 **DESCRIPTION**8277 The <ftw.h> header shall define the **FTW** structure that includes at least the following members:

8278 int base

8279 int level

8280 The <ftw.h> header shall define macros for use as values of the third argument to the
8281 application-supplied function that is passed as the second argument to *ftw()* and *nftw()*:

8282 FTW_F File.

8283 FTW_D Directory.

8284 FTW_DNR Directory without read permission.

8285 FTW_DP Directory with subdirectories visited.

8286 FTW_NS Unknown type; *stat()* failed.

8287 FTW_SL Symbolic link.

8288 FTW_SLN Symbolic link that names a nonexistent file.

8289 The <ftw.h> header shall define macros for use as values of the fourth argument to *nftw()*:8290 FTW_PHYS Physical walk, does not follow symbolic links. Otherwise, *nftw()* follows
8291 links but does not walk down any path that crosses itself.

8292 FTW_MOUNT The walk does not cross a mount point.

8293 FTW_DEPTH All subdirectories are visited before the directory itself.

8294 FTW_CHDIR The walk changes to each directory before reading it.

8295 The following shall be declared as functions and may also be defined as macros. Function
8296 prototypes shall be provided.8297 int ftw(const char *, int (*)(const char *, const struct stat *,
8298 int), int);8299 int nftw(const char *, int (*)(const char *, const struct stat *,
8300 int, struct FTW*), int, int);8301 The <ftw.h> header shall define the **stat** structure and the symbolic names for *st_mode* and the
8302 file type test macros as described in <sys/stat.h>.

8303 Inclusion of the <ftw.h> header may also make visible all symbols from <sys/stat.h>.

8304 **APPLICATION USAGE**

8305 None.

8306 **RATIONALE**

8307 None.

8308 **FUTURE DIRECTIONS**

8309 None.

8310 **SEE ALSO**

8311 <sys/stat.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *ftw()*, *nftw()*

8312 **CHANGE HISTORY**

8313 First released in Issue 1. Derived from Issue 1 of the SVID.

8314 **Issue 5**

8315 A description of FTW_DP is added.

8316 **NAME**

8317 glob.h — pathname pattern-matching types

8318 **SYNOPSIS**

8319 #include <glob.h>

8320 **DESCRIPTION**8321 The <glob.h> header shall define the structures and symbolic constants used by the *glob()*
8322 function.8323 The structure type **glob_t** shall contain at least the following members:8324 size_t gl_pathc Count of paths matched by *pattern*.

8325 char **gl_pathv Pointer to a list of matched pathnames.

8326 size_t gl_offs Slots to reserve at the beginning of *gl_pathv*.8327 The following constants shall be provided as values for the *flags* argument:

8328 GLOB_APPEND Append generated pathnames to those previously obtained.

8329 GLOB_DOOFFS Specify how many null pointers to add to the beginning of *gl_pathv*.8330 GLOB_ERR Cause *glob()* to return on error.8331 GLOB_MARK Each pathname that is a directory that matches *pattern* has a slash
8332 appended.8333 GLOB_NOCHECK If *pattern* does not match any pathname, then return a list consisting of
8334 only *pattern*.

8335 GLOB_NOESCAPE Disable backslash escaping.

8336 GLOB_NOSORT Do not sort the pathnames returned.

8337 The following constants shall be defined as error return values:

8338 GLOB_ABORTED The scan was stopped because GLOB_ERR was set or *(*errfunc)()*
8339 returned non-zero.8340 GLOB_NOMATCH The pattern does not match any existing pathname, and
8341 GLOB_NOCHECK was not set in *flags*.

8342 GLOB_NOSPACE An attempt to allocate memory failed.

8343 OB XSI GLOB_NOSYS Reserved.

8344 The following shall be declared as functions and may also be defined as macros. Function
8345 prototypes shall be provided.8346 int glob(const char *restrict, int, int (*)(const char *, int),
8347 glob_t *restrict);

8348 void globfree (glob_t *);

8349 The implementation may define additional macros or constants using names beginning with
8350 GLOB_.

8351 **APPLICATION USAGE**

8352 None.

8353 **RATIONALE**

8354 None.

8355 **FUTURE DIRECTIONS**

8356 None.

8357 **SEE ALSO**

8358 The System Interfaces volume of IEEE Std 1003.1-2001, *glob()*, the Shell and Utilities volume of
8359 IEEE Std 1003.1-2001

8360 **CHANGE HISTORY**

8361 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8362 **Issue 6**8363 The **restrict** keyword is added to the prototype for *glob()*.

8364 The constant GLOB_NOSYS is marked obsolescent.

8365 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/8 is applied, correcting the *glob()* |
8366 prototype definition by removing the **restrict** qualifier from the function pointer argument. |

8367 **NAME**

8368 grp.h — group structure

8369 **SYNOPSIS**

8370 #include <grp.h>

8371 **DESCRIPTION**8372 The <grp.h> header shall declare the structure **group** which shall include the following
8373 members:

8374 char *gr_name The name of the group.
 8375 gid_t gr_gid Numerical group ID.
 8376 char **gr_mem Pointer to a null-terminated array of character
 8377 pointers to member names.

8378 The **gid_t** type shall be defined as described in <sys/types.h>.8379 The following shall be declared as functions and may also be defined as macros. Function
8380 prototypes shall be provided.

```

8381       struct group  *getgrgid(gid_t);
8382       struct group  *getgrnam(const char *);
8383 TSF       int        getgrgid_r(gid_t, struct group *, char *,
8384                               size_t, struct group **);
8385       int        getgrnam_r(const char *, struct group *, char *,
8386                               size_t, struct group **);
8387 XSI       struct group *getgrent(void);
8388       void       endgrent(void);
8389       void       setgrent(void);

```

8390

8391 **APPLICATION USAGE**

8392 None.

8393 **RATIONALE**

8394 None.

8395 **FUTURE DIRECTIONS**

8396 None.

8397 **SEE ALSO**

8398 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *endgrent()*, *getgrgid()*,
 8399 *getgrnam()*

8400 **CHANGE HISTORY**

8401 First released in Issue 1.

8402 **Issue 5**

8403 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8404 **Issue 6**8405 The following new requirements on POSIX implementations derive from alignment with the
8406 Single UNIX Specification:

- 8407
 - The definition of **gid_t** is mandated.
 - The *getgrgid_r()* and *getgrnam_r()* functions are marked as part of the Thread-Safe Functions option.
- 8408
8409

8410 **NAME**

8411 iconv.h — codeset conversion facility

8412 **SYNOPSIS**

8413 XSI `#include <iconv.h>`

8414

8415 **DESCRIPTION**

8416 The <iconv.h> header shall define the following type:

8417 **iconv_t** Identifies the conversion from one codeset to another.

8418 The following shall be declared as functions and may also be defined as macros. Function
8419 prototypes shall be provided.

```
8420 iconv_t iconv_open(const char *, const char *);
8421 size_t iconv(iconv_t, char **restrict, size_t *restrict,
8422             char **restrict, size_t *restrict);
8423 int iconv_close(iconv_t);
```

8424 **APPLICATION USAGE**

8425 None.

8426 **RATIONALE**

8427 None.

8428 **FUTURE DIRECTIONS**

8429 None.

8430 **SEE ALSO**

8431 The System Interfaces volume of IEEE Std 1003.1-2001, *iconv()*, *iconv_close()*, *iconv_open()*

8432 **CHANGE HISTORY**

8433 First released in Issue 4.

8434 **Issue 6**

8435 The **restrict** keyword is added to the prototype for *iconv()*.

8436 **NAME**

8437 inttypes.h — fixed size integer types

8438 **SYNOPSIS**

8439 #include <inttypes.h>

8440 **DESCRIPTION**

8441 **CX** Some of the functionality described on this reference page extends the ISO C standard.
 8442 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 8443 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 8444 symbols in this header.

8445 The <inttypes.h> header shall include the <stdint.h> header.

8446 The <inttypes.h> header shall include a definition of at least the following type:

8447 **imaxdiv_t** Structure type that is the type of the value returned by the *imaxdiv()* function.

8448 The following macros shall be defined. Each expands to a character string literal containing a
 8449 conversion specifier, possibly modified by a length modifier, suitable for use within the *format*
 8450 argument of a formatted input/output function when converting the corresponding integer
 8451 type. These macros have the general form of PRI (character string literals for the *fprintf()* and
 8452 *fwprintf()* family of functions) or SCN (character string literals for the *fscanf()* and *fwscanf()*
 8453 family of functions), followed by the conversion specifier, followed by a name corresponding to
 8454 a similar type name in <stdint.h>. In these names, *N* represents the width of the type as
 8455 described in <stdint.h>. For example, *PRIdFAST32* can be used in a format string to print the
 8456 value of an integer of type **int_fast32_t**.

8457 The *fprintf()* macros for signed integers are:

8458	PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
8459	PRiN	PRiLEASTN	PRiFASTN	PRiMAX	PRiPTR

8460 The *fprintf()* macros for unsigned integers are:

8461	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
8462	PRiN	PRiLEASTN	PRiFASTN	PRiMAX	PRiPTR
8463	PRIxN	PRIxLEASTN	PRIxFASTN	PRIxMAX	PRIxPTR
8464	PRIXN	PRIXLEASTN	PRIXFASTN	PRIXMAX	PRIXPTR

8465 The *fscanf()* macros for signed integers are:

8466	SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
8467	SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

8468 The *fscanf()* macros for unsigned integers are:

8469	SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
8470	SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
8471	SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

8472 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* and
 8473 *fwprintf()* macros shall be defined and the corresponding *fscanf()* and *fwscanf()* macros shall be
 8474 defined unless the implementation does not have a suitable modifier for the type.

8475 The following shall be declared as functions and may also be defined as macros. Function
 8476 prototypes shall be provided.

```
8477 intmax_t imaxabs(intmax_t);
8478 imaxdiv_t imaxdiv(intmax_t, intmax_t);
8479 intmax_t strtoumax(const char *restrict, char **restrict, int);
```



```

8480     uintmax_t strtoumax(const char *restrict, char **restrict, int);
8481     intmax_t wcstoimax(const wchar_t *restrict, wchar_t **restrict, int);
8482     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);

```

8483 EXAMPLES

```

8484     #include <inttypes.h>
8485     #include <wchar.h>
8486     int main(void)
8487     {
8488         uintmax_t i = UINTMAX_MAX; // This type always exists.
8489         wprintf(L"The largest integer value is %020"
8490             PRIxMAX "\n", i);
8491         return 0;
8492     }

```

8493 APPLICATION USAGE

8494 The purpose of <inttypes.h> is to provide a set of integer types whose definitions are consistent
8495 across machines and independent of operating systems and other implementation
8496 idiosyncrasies. It defines, via **typedef**, integer types of various sizes. Implementations are free to
8497 **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of
8498 this header will greatly increase the portability of applications across platforms.

8499 RATIONALE

8500 The ISO/IEC 9899:1990 standard specified that the language should support four signed and
8501 unsigned integer data types—**char**, **short**, **int**, and **long**—but placed very little requirement on
8502 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and
8503 not smaller than 32 bits. For 16-bit systems, most implementations assigned 8, 16, 16, and 32 bits
8504 to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice has been to
8505 assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems
8506 for users who migrate from one system to another which assigns different sizes to integer types,
8507 because the ISO C standard integer promotion rule can produce silent changes unexpectedly.
8508 The need for defining an extended integer type increased with the introduction of 64-bit
8509 systems.

8510 FUTURE DIRECTIONS

8511 Macro names beginning with PRI or SCN followed by any lowercase letter or 'X' may be added
8512 to the macros defined in the <inttypes.h> header.

8513 SEE ALSO

8514 The System Interfaces volume of IEEE Std 1003.1-2001, *imaxdiv()*

8515 CHANGE HISTORY

8516 First released in Issue 5.

8517 Issue 6

8518 The Open Group Base Resolution bwg97-006 is applied.

8519 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

8520 **NAME**

8521 iso646.h — alternative spellings

8522 **SYNOPSIS**

8523 #include <iso646.h>

8524 **DESCRIPTION**

8525 cx The functionality described on this reference page is aligned with the ISO C standard. Any
8526 conflict between the requirements described here and the ISO C standard is unintentional. This
8527 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8528 The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the
8529 corresponding tokens (on the right):

8530 and &&

8531 and_eq &=

8532 bitand &

8533 bitor |

8534 compl ~

8535 not !

8536 not_eq !=

8537 or | |

8538 or_eq |=

8539 xor ^

8540 xor_eq ^=

8541 **APPLICATION USAGE**

8542 None.

8543 **RATIONALE**

8544 None.

8545 **FUTURE DIRECTIONS**

8546 None.

8547 **SEE ALSO**

8548 None.

8549 **CHANGE HISTORY**

8550 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

8551 **NAME**

8552 langinfo.h — language information constants

8553 **SYNOPSIS**

8554 xSI #include <langinfo.h>

8555

8556 **DESCRIPTION**

8557 The <langinfo.h> header contains the constants used to identify items of *langinfo* data (see
8558 *nl_langinfo()*). The type of the constant, **nl_item**, shall be defined as described in <nl_types.h>.

8559 The following constants shall be defined. The entries under **Category** indicate in which
8560 *setlocale()* category each item is defined.

8561

Constant	Category	Meaning
CODESET	LC_CTYPE	Codeset name.
D_T_FMT	LC_TIME	String for formatting date and time.
D_FMT	LC_TIME	Date format string.
T_FMT	LC_TIME	Time format string.
T_FMT_AMPM	LC_TIME	a.m. or p.m. time format string.
AM_STR	LC_TIME	Ante-meridiem affix.
PM_STR	LC_TIME	Post-meridiem affix.
DAY_1	LC_TIME	Name of the first day of the week (for example, Sunday).
DAY_2	LC_TIME	Name of the second day of the week (for example, Monday).
DAY_3	LC_TIME	Name of the third day of the week (for example, Tuesday).
DAY_4	LC_TIME	Name of the fourth day of the week (for example, Wednesday).
DAY_5	LC_TIME	Name of the fifth day of the week (for example, Thursday).
DAY_6	LC_TIME	Name of the sixth day of the week (for example, Friday).
DAY_7	LC_TIME	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	LC_TIME	Abbreviated name of the first day of the week.
ABDAY_2	LC_TIME	Abbreviated name of the second day of the week.
ABDAY_3	LC_TIME	Abbreviated name of the third day of the week.
ABDAY_4	LC_TIME	Abbreviated name of the fourth day of the week.
ABDAY_5	LC_TIME	Abbreviated name of the fifth day of the week.
ABDAY_6	LC_TIME	Abbreviated name of the sixth day of the week.
ABDAY_7	LC_TIME	Abbreviated name of the seventh day of the week.
MON_1	LC_TIME	Name of the first month of the year.
MON_2	LC_TIME	Name of the second month.
MON_3	LC_TIME	Name of the third month.
MON_4	LC_TIME	Name of the fourth month.
MON_5	LC_TIME	Name of the fifth month.
MON_6	LC_TIME	Name of the sixth month.
MON_7	LC_TIME	Name of the seventh month.
MON_8	LC_TIME	Name of the eighth month.
MON_9	LC_TIME	Name of the ninth month.
MON_10	LC_TIME	Name of the tenth month.
MON_11	LC_TIME	Name of the eleventh month.
MON_12	LC_TIME	Name of the twelfth month.

8598

8599

8600

8601

8602

8603

8604

8605

8606

8607

8608

8609

8610

8611

8612

8613

8614

8615

8616

8617

8618

8619

8620

8621

8622

8623

8624

8625

Constant	Category	Meaning
ABMON_1	LC_TIME	Abbreviated name of the first month.
ABMON_2	LC_TIME	Abbreviated name of the second month.
ABMON_3	LC_TIME	Abbreviated name of the third month.
ABMON_4	LC_TIME	Abbreviated name of the fourth month.
ABMON_5	LC_TIME	Abbreviated name of the fifth month.
ABMON_6	LC_TIME	Abbreviated name of the sixth month.
ABMON_7	LC_TIME	Abbreviated name of the seventh month.
ABMON_8	LC_TIME	Abbreviated name of the eighth month.
ABMON_9	LC_TIME	Abbreviated name of the ninth month.
ABMON_10	LC_TIME	Abbreviated name of the tenth month.
ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
ERA	LC_TIME	Era description segments.
ERA_D_FMT	LC_TIME	Era date format string.
ERA_D_T_FMT	LC_TIME	Era date and time format string.
ERA_T_FMT	LC_TIME	Era time format string.
ALT_DIGITS	LC_TIME	Alternative symbols for digits.
RADIXCHAR	LC_NUMERIC	Radix character.
THOUSEP	LC_NUMERIC	Separator for thousands.
YESEXPR	LC_MESSAGES	Affirmative response expression.
NOEXPR	LC_MESSAGES	Negative response expression.
CRNCYSTR	LC_MONETARY	Local currency symbol, preceded by '–' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character. If the local currency symbol is the empty string, implementations may return the empty string ("").

8626

8627

If the locale's values for **p_cs_precedes** and **n_cs_precedes** do not match, the value of *nl_langinfo*(CRNCYSTR) is unspecified.

8628

8629

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

8630

```
char *nl_langinfo(nl_item);
```

8631

Inclusion of the <langinfo.h> header may also make visible all symbols from <nl_types.h>.

8632

APPLICATION USAGE

8633

8634

8635

8636

8637

Wherever possible, users are advised to use functions compatible with those in the ISO C standard to access items of *langinfo* data. In particular, the *strftime*() function should be used to access date and time information defined in category *LC_TIME*. The *localeconv*() function should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and *CRNCYSTR*.

8638

RATIONALE

8639

None.

8640

FUTURE DIRECTIONS

8641

None.

8642

SEE ALSO

8643

8644

The System Interfaces volume of IEEE Std 1003.1-2001, *nl_langinfo*(), *localeconv*(), *strfmon*(), *strftime*(), Chapter 7 (on page 123)

8645 **CHANGE HISTORY**

8646 First released in Issue 2.

8647 **Issue 5**

8648 The constants YESSTR and NOSTR are marked LEGACY.

8649 **Issue 6**

8650 The constants YESSTR and NOSTR are removed.

8651 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/9 is applied, adding a sentence to
 8652 the“Meaning” column entry for the CRNCYSTR constant. This change is to accommodate
 8653 historic practice.

8654 **NAME**
8655 libgen.h — definitions for pattern matching functions

8656 **SYNOPSIS**
8657 xSI `#include <libgen.h>`
8658

8659 **DESCRIPTION**
8660 The following shall be declared as functions and may also be defined as macros. Function
8661 prototypes shall be provided.
8662 `char *basename(char *);`
8663 `char *dirname(char *);`

8664 **APPLICATION USAGE**
8665 None.

8666 **RATIONALE**
8667 None.

8668 **FUTURE DIRECTIONS**
8669 None.

8670 **SEE ALSO**
8671 The System Interfaces volume of IEEE Std 1003.1-2001, *basename()*, *dirname()*

8672 **CHANGE HISTORY**
8673 First released in Issue 4, Version 2.

8674 **Issue 5**
8675 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first
8676 argument is of type **char *** rather than **const char ***.

8677 **Issue 6**
8678 The **__loc1** symbol and the *regcmp()* and *regex()* functions are removed.

8679 **NAME**

8680 limits.h — implementation-defined constants

8681 **SYNOPSIS**

8682 #include <limits.h>

8683 **DESCRIPTION**

8684 CX Some of the functionality described on this reference page extends the ISO C standard.
8685 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
8686 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
8687 symbols in this header.

8688 CX Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such
8689 symbols are not shown as CX shaded.

8690 The <limits.h> header shall define various symbolic names. Different categories of names are
8691 described below.

8692 The names represent various limits on resources that the implementation imposes on
8693 applications.

8694 Implementations may choose any appropriate value for each limit, provided it is not more
8695 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names
8696 beginning with _POSIX may be found in <unistd.h>.

8697 Applications should not assume any particular value for a limit. To achieve maximum
8698 portability, an application should not require more resource than the Minimum Acceptable
8699 Value quantity. However, an application wishing to avail itself of the full amount of a resource
8700 available on an implementation may make use of the value given in <limits.h> on that
8701 particular implementation, by using the symbolic names listed below. It should be noted,
8702 however, that many of the listed limits are not invariant, and at runtime, the value of the limit
8703 may differ from those given in this header, for the following reasons:

- 8704 • The limit is pathname-dependent.
- 8705 • The limit differs between the compile and runtime machines.

8706 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to
8707 determine the actual value of a limit at runtime.

8708 The items in the list ending in _MIN give the most negative values that the mathematical types
8709 are guaranteed to be capable of representing. Numbers of a more negative value may be
8710 supported on some implementations, as indicated by the <limits.h> header on the
8711 implementation, but applications requiring such numbers are not guaranteed to be portable to
8712 all implementations. For positive constants ending in _MIN, this indicates the minimum
8713 acceptable value.

8714 **Runtime Invariant Values (Possibly Indeterminate)**

8715 A definition of one of the symbolic names in the following list shall be omitted from <limits.h>
8716 on specific implementations where the corresponding value is equal to or greater than the stated
8717 minimum, but is unspecified.

8718 This indetermination might depend on the amount of available memory space on a specific
8719 instance of a specific implementation. The actual value supported by a specific instance shall be
8720 provided by the *sysconf()* function.

8721 AIO {AIO_LISTIO_MAX}
8722 Maximum number of I/O operations in a single list I/O call supported by the

8723		implementation.
8724		Minimum Acceptable Value: {_POSIX_AIO_LISTIO_MAX}
8725	AIO	{AIO_MAX}
8726		Maximum number of outstanding asynchronous I/O operations supported by the
8727		implementation.
8728		Minimum Acceptable Value: {_POSIX_AIO_MAX}
8729	AIO	{AIO_PRIO_DELTA_MAX}
8730		The maximum amount by which a process can decrease its asynchronous I/O priority level
8731		from its own scheduling priority.
8732		Minimum Acceptable Value: 0
8733		{ARG_MAX}
8734		Maximum length of argument to the <i>exec</i> functions including environment data.
8735		Minimum Acceptable Value: {_POSIX_ARG_MAX}
8736	XSI	{ATEXIT_MAX}
8737		Maximum number of functions that may be registered with <i>atexit()</i> .
8738		Minimum Acceptable Value: 32
8739		{CHILD_MAX}
8740		Maximum number of simultaneous processes per real user ID.
8741		Minimum Acceptable Value: {_POSIX_CHILD_MAX}
8742	TMR	{DELAYTIMER_MAX}
8743		Maximum number of timer expiration overruns.
8744		Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}
8745		{HOST_NAME_MAX}
8746		Maximum length of a host name (not including the terminating null) as returned from the
8747		<i>gethostname()</i> function.
8748		Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}
8749	XSI	{IOV_MAX}
8750		Maximum number of iovec structures that one process has available for use with <i>readv()</i> or
8751		<i>writev()</i> .
8752		Minimum Acceptable Value: {_XOPEN_IOV_MAX}
8753		{LOGIN_NAME_MAX}
8754		Maximum length of a login name.
8755		Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}
8756	MSG	{MQ_OPEN_MAX}
8757		The maximum number of open message queue descriptors a process may hold.
8758		Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}
8759	MSG	{MQ_PRIO_MAX}
8760		The maximum number of message priorities supported by the implementation.
8761		Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}
8762		{OPEN_MAX}
8763		Maximum number of files that one process can have open at any one time.
8764		Minimum Acceptable Value: {_POSIX_OPEN_MAX}
8765		{PAGESIZE}
8766		Size in bytes of a page.
8767		Minimum Acceptable Value: 1

8768	XSI	{PAGE_SIZE}
8769		Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is
8770		defined with the same value.
8771	THR	{PTHREAD_DESTRUCTOR_ITERATIONS}
8772		Maximum number of attempts made to destroy a thread's thread-specific data values on
8773		thread exit.
8774		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8775	THR	{PTHREAD_KEYS_MAX}
8776		Maximum number of data keys that can be created by a process.
8777		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
8778	THR	{PTHREAD_STACK_MIN}
8779		Minimum size in bytes of thread stack storage.
8780		Minimum Acceptable Value: 0
8781	THR	{PTHREAD_THREADS_MAX}
8782		Maximum number of threads that can be created per process.
8783		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
8784		{RE_DUP_MAX}
8785		The number of repeated occurrences of a BRE permitted by the <i>regex</i> (<i>c</i>) and <i>regcomp</i> (<i>c</i>)
8786		functions when using the interval notation <i>{(m,n)}</i> ; see Section 9.3.6 (on page 174).
8787		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
8788	RTS	{RTSIG_MAX}
8789		Maximum number of realtime signals reserved for application use in this implementation.
8790		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
8791	SEM	{SEM_NSEMS_MAX}
8792		Maximum number of semaphores that a process may have.
8793		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
8794	SEM	{SEM_VALUE_MAX}
8795		The maximum value a semaphore may have.
8796		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
8797	RTS	{SIGQUEUE_MAX}
8798		Maximum number of queued signals that a process may send and have pending at the
8799		receiver(s) at any time.
8800		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
8801	SS TSP	{SS_REPL_MAX}
8802		The maximum number of replenishment operations that may be simultaneously pending
8803		for a particular sporadic server scheduler.
8804		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}
8805		{STREAM_MAX}
8806		The number of streams that one process can have open at one time. If defined, it has the
8807		same value as {FOPEN_MAX} (see <stdio.h>).
8808		Minimum Acceptable Value: {_POSIX_STREAM_MAX}
8809		{SYMLOOP_MAX}
8810		Maximum number of symbolic links that can be reliably traversed in the resolution of a
8811		pathname in the absence of a loop.
8812		Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}

8813	TMR	{TIMER_MAX}
8814		Maximum number of timers per process supported by the implementation.
8815		Minimum Acceptable Value: {_POSIX_TIMER_MAX}
8816	TRC	{TRACE_EVENT_NAME_MAX}
8817		Maximum length of the trace event name.
8818		Minimum Acceptable Value: {_POSIX_TRACE_EVENT_NAME_MAX}
8819	TRC	{TRACE_NAME_MAX}
8820		Maximum length of the trace generation version string or of the trace stream name.
8821		Minimum Acceptable Value: {_POSIX_TRACE_NAME_MAX}
8822	TRC	{TRACE_SYS_MAX}
8823		Maximum number of trace streams that may simultaneously exist in the system.
8824		Minimum Acceptable Value: {_POSIX_TRACE_SYS_MAX}
8825	TRC	{TRACE_USER_EVENT_MAX}
8826		Maximum number of user trace event type identifiers that may simultaneously exist in a
8827		traced process, including the predefined user trace event
8828		POSIX_TRACE_UNNAMED_USER_EVENT.
8829		Minimum Acceptable Value: {_POSIX_TRACE_USER_EVENT_MAX}
8830		{TTY_NAME_MAX}
8831		Maximum length of terminal device name.
8832		Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}
8833		{TZNAME_MAX}
8834		Maximum number of bytes supported for the name of a timezone (not of the <i>TZ</i> variable).
8835		Minimum Acceptable Value: {_POSIX_TZNAME_MAX}
8836	Note:	The length given by {TZNAME_MAX} does not include the quoting characters mentioned in
8837		Section 8.3 (on page 165).

8838 Pathname Variable Values

8839 The values in the following list may be constants within an implementation or may vary from
 8840 one pathname to another. For example, file systems or directories may have different
 8841 characteristics.

8842 A definition of one of the values shall be omitted from the <limits.h> header on specific
 8843 implementations where the corresponding value is equal to or greater than the stated minimum,
 8844 but where the value can vary depending on the file to which it is applied. The actual value
 8845 supported for a specific pathname shall be provided by the *pathconf()* function.

8846 {FILESIZEBITS}

8847 Minimum number of bits needed to represent, as a signed integer value, the maximum size
 8848 of a regular file allowed in the specified directory.

8849 Minimum Acceptable Value: 32

8850 {LINK_MAX}

8851 Maximum number of links to a single file.

8852 Minimum Acceptable Value: {_POSIX_LINK_MAX}

8853 {MAX_CANON}

8854 Maximum number of bytes in a terminal canonical input line.

8855 Minimum Acceptable Value: {_POSIX_MAX_CANON}

8856 {MAX_INPUT}

8857 Minimum number of bytes for which space is available in a terminal input queue; therefore,

8858 the maximum number of bytes a conforming application may require to be typed as input
 8859 before reading them.
 8860 Minimum Acceptable Value: {_POSIX_MAX_INPUT}

8861 {NAME_MAX}
 8862 Maximum number of bytes in a filename (not including terminating null).
 8863 Minimum Acceptable Value: {_POSIX_NAME_MAX}
 8864 XSI Minimum Acceptable Value: {_XOPEN_NAME_MAX}

8865 {PATH_MAX}
 8866 Maximum number of bytes in a pathname, including the terminating null character.
 8867 Minimum Acceptable Value: {_POSIX_PATH_MAX}
 8868 XSI Minimum Acceptable Value: {_XOPEN_PATH_MAX}

8869 {PIPE_BUF}
 8870 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
 8871 Minimum Acceptable Value: {_POSIX_PIPE_BUF}

8872 ADV {POSIX_ALLOC_SIZE_MIN}
 8873 Minimum number of bytes of storage actually allocated for any portion of a file.
 8874 Minimum Acceptable Value: Not specified.

8875 ADV {POSIX_REC_INCR_XFER_SIZE}
 8876 Recommended increment for file transfer sizes between the
 8877 {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
 8878 Minimum Acceptable Value: Not specified.

8879 ADV {POSIX_REC_MAX_XFER_SIZE}
 8880 Maximum recommended file transfer size.
 8881 Minimum Acceptable Value: Not specified.

8882 ADV {POSIX_REC_MIN_XFER_SIZE}
 8883 Minimum recommended file transfer size.
 8884 Minimum Acceptable Value: Not specified.

8885 ADV {POSIX_REC_XFER_ALIGN}
 8886 Recommended file transfer buffer alignment.
 8887 Minimum Acceptable Value: Not specified.

8888 {SYMLINK_MAX}
 8889 Maximum number of bytes in a symbolic link.
 8890 Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

Runtime Increaseable Values

8891 The magnitude limitations in the following list shall be fixed by specific implementations. An
 8892 application should assume that the value supplied by <limits.h> in a specific implementation is
 8893 the minimum that pertains whenever the application is run under that implementation. A
 8894 specific instance of a specific implementation may increase the value relative to that supplied by
 8895 <limits.h> for that implementation. The actual value supported by a specific instance shall be
 8896 provided by the *sysconf()* function.

8897

8898 {BC_BASE_MAX}
 8899 Maximum *obase* values allowed by the *bc* utility.
 8900 Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}

8901 {BC_DIM_MAX}
 8902 Maximum number of elements permitted in an array by the *bc* utility.

8903 Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}
 8904 {BC_SCALE_MAX}
 8905 Maximum *scale* value allowed by the *bc* utility.
 8906 Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}
 8907 {BC_STRING_MAX}
 8908 Maximum length of a string constant accepted by the *bc* utility.
 8909 Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}
 8910 {CHARCLASS_NAME_MAX}
 8911 Maximum number of bytes in a character class name.
 8912 Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}
 8913 {COLL_WEIGHTS_MAX}
 8914 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
 8915 keyword in the locale definition file; see Chapter 7 (on page 123).
 8916 Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}
 8917 {EXPR_NEST_MAX}
 8918 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
 8919 Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}
 8920 {LINE_MAX}
 8921 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
 8922 standard input or another file), when the utility is described as processing text files. The
 8923 length includes room for the trailing <newline>.
 8924 Minimum Acceptable Value: {_POSIX2_LINE_MAX}
 8925 {NGROUPS_MAX}
 8926 Maximum number of simultaneous supplementary group IDs per process.
 8927 Minimum Acceptable Value: {_POSIX_NGROUPS_MAX}
 8928 {RE_DUP_MAX}
 8929 Maximum number of repeated occurrences of a regular expression permitted when using
 8930 the interval notation `\{m,n\}`; see Chapter 9 (on page 169).
 8931 Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}

8932 Maximum Values

8933 TMR The symbolic constants in the following list shall be defined in <limits.h> with the values
 8934 shown. These are symbolic names for the most restrictive value for certain features on an
 8935 implementation supporting the Timers option. A conforming implementation shall provide
 8936 values no larger than these values. A conforming application must not require a smaller value
 8937 for correct operation.
 8938 TMR {_POSIX_CLOCKRES_MIN}
 8939 The resolution of the *CLOCK_REALTIME* clock, in nanoseconds.
 8940 Value: 20 000 000
 8941 MON If the Monotonic Clock option is supported, the resolution of the *CLOCK_MONOTONIC*
 8942 clock, in nanoseconds, is represented by {_POSIX_CLOCKRES_MIN}.

8943 **Minimum Values**

8944 The symbolic constants in the following list shall be defined in <limits.h> with the values
 8945 shown. These are symbolic names for the most restrictive value for certain features on an
 8946 implementation conforming to this volume of IEEE Std 1003.1-2001. Related symbolic constants
 8947 are defined elsewhere in this volume of IEEE Std 1003.1-2001 which reflect the actual
 8948 implementation and which need not be as restrictive. A conforming implementation shall
 8949 provide values at least this large. A strictly conforming application must not require a larger
 8950 value for correct operation.

8951 AIO { _POSIX_AIO_LISTIO_MAX}
 8952 The number of I/O operations that can be specified in a list I/O call.
 8953 Value: 2

8954 AIO { _POSIX_AIO_MAX}
 8955 The number of outstanding asynchronous I/O operations.
 8956 Value: 1

8957 { _POSIX_ARG_MAX}
 8958 Maximum length of argument to the *exec* functions including environment data.
 8959 Value: 4 096

8960 { _POSIX_CHILD_MAX}
 8961 Maximum number of simultaneous processes per real user ID.
 8962 Value: 25

8963 TMR { _POSIX_DELAYTIMER_MAX}
 8964 The number of timer expiration overruns.
 8965 Value: 32

8966 { _POSIX_HOST_NAME_MAX}
 8967 Maximum length of a host name (not including the terminating null) as returned from the
 8968 *gethostname()* function.
 8969 Value: 255

8970 { _POSIX_LINK_MAX}
 8971 Maximum number of links to a single file.
 8972 Value: 8

8973 { _POSIX_LOGIN_NAME_MAX}
 8974 The size of the storage required for a login name, in bytes, including the terminating null.
 8975 Value: 9

8976 { _POSIX_MAX_CANON}
 8977 Maximum number of bytes in a terminal canonical input queue.
 8978 Value: 255

8979 { _POSIX_MAX_INPUT}
 8980 Maximum number of bytes allowed in a terminal input queue.
 8981 Value: 255

8982 MSG { _POSIX_MQ_OPEN_MAX}
 8983 The number of message queues that can be open for a single process.
 8984 Value: 8

8985 MSG { _POSIX_MQ_PRIO_MAX}
 8986 The maximum number of message priorities supported by the implementation.
 8987 Value: 32

8988		{_POSIX_NAME_MAX}
8989		Maximum number of bytes in a filename (not including terminating null).
8990		Value: 14
8991		{_POSIX_NGROUPS_MAX}
8992		Maximum number of simultaneous supplementary group IDs per process.
8993		Value: 8
8994		{_POSIX_OPEN_MAX}
8995		Maximum number of files that one process can have open at any one time.
8996		Value: 20
8997		{_POSIX_PATH_MAX}
8998		Maximum number of bytes in a pathname.
8999		Value: 256
9000		{_POSIX_PIPE_BUF}
9001		Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
9002		Value: 512
9003		{_POSIX_RE_DUP_MAX}
9004		The number of repeated occurrences of a BRE permitted by the <i>regex</i> (<i>re</i>) and <i>regcomp</i> (<i>re</i>)
9005		functions when using the interval notation <i>{(m,n)}</i> ; see Section 9.3.6 (on page 174).
9006		Value: 255
9007	RTS	{_POSIX_RTSIG_MAX}
9008		The number of realtime signal numbers reserved for application use.
9009		Value: 8
9010	SEM	{_POSIX_SEM_NSEMS_MAX}
9011		The number of semaphores that a process may have.
9012		Value: 256
9013	SEM	{_POSIX_SEM_VALUE_MAX}
9014		The maximum value a semaphore may have.
9015		Value: 32 767
9016	RTS	{_POSIX_SIGQUEUE_MAX}
9017		The number of queued signals that a process may send and have pending at the receiver(s)
9018		at any time.
9019		Value: 32
9020		{_POSIX_SSIZE_MAX}
9021		The value that can be stored in an object of type ssize_t .
9022		Value: 32 767
9023		{_POSIX_STREAM_MAX}
9024		The number of streams that one process can have open at one time.
9025		Value: 8
9026	SS TSP	{_POSIX_SS_REPL_MAX}
9027		The number of replenishment operations that may be simultaneously pending for a
9028		particular sporadic server scheduler.
9029		Value: 4
9030		{_POSIX_SYMLINK_MAX}
9031		The number of bytes in a symbolic link.
9032		Value: 255

9033		{_POSIX_SYMLOOP_MAX}
9034		The number of symbolic links that can be traversed in the resolution of a pathname in the
9035		absence of a loop.
9036		Value: 8
9037	THR	{_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
9038		The number of attempts made to destroy a thread's thread-specific data values on thread
9039		exit.
9040		Value: 4
9041	THR	{_POSIX_THREAD_KEYS_MAX}
9042		The number of data keys per process.
9043		Value: 128
9044	THR	{_POSIX_THREAD_THREADS_MAX}
9045		The number of threads per process.
9046		Value: 64
9047	TMR	{_POSIX_TIMER_MAX}
9048		The per-process number of timers.
9049		Value: 32
9050	TRC	{_POSIX_TRACE_EVENT_NAME_MAX}
9051		The length in bytes of a trace event name.
9052		Value: 30
9053	TRC	{_POSIX_TRACE_NAME_MAX}
9054		The length in bytes of a trace generation version string or a trace stream name.
9055		Value: 8
9056	TRC	{_POSIX_TRACE_SYS_MAX}
9057		The number of trace streams that may simultaneously exist in the system.
9058		Value: 8
9059	TRC	{_POSIX_TRACE_USER_EVENT_MAX}
9060		The number of user trace event type identifiers that may simultaneously exist in a traced
9061		process, including the predefined user trace event
9062		POSIX_TRACE_UNNAMED_USER_EVENT.
9063		Value: 32
9064		{_POSIX_TTY_NAME_MAX}
9065		The size of the storage required for a terminal device name, in bytes, including the
9066		terminating null.
9067		Value: 9
9068		{_POSIX_TZNAME_MAX}
9069		Maximum number of bytes supported for the name of a timezone (not of the <i>TZ</i> variable).
9070		Value: 6
9071		Note: The length given by {_POSIX_TZNAME_MAX} does not include the quoting characters
9072		mentioned in Section 8.3 (on page 165).
9073		{_POSIX2_BC_BASE_MAX}
9074		Maximum <i>obase</i> values allowed by the <i>bc</i> utility.
9075		Value: 99
9076		{_POSIX2_BC_DIM_MAX}
9077		Maximum number of elements permitted in an array by the <i>bc</i> utility.
9078		Value: 2 048

```

9079      {_POSIX2_BC_SCALE_MAX}
9080          Maximum scale value allowed by the bc utility.
9081          Value: 99

9082      {_POSIX2_BC_STRING_MAX}
9083          Maximum length of a string constant accepted by the bc utility.
9084          Value: 1 000

9085      {_POSIX2_CHARCLASS_NAME_MAX}
9086          Maximum number of bytes in a character class name.
9087          Value: 14

9088      {_POSIX2_COLL_WEIGHTS_MAX}
9089          Maximum number of weights that can be assigned to an entry of the LC_COLLATE order
9090          keyword in the locale definition file; see Chapter 7 (on page 123).
9091          Value: 2

9092      {_POSIX2_EXPR_NEST_MAX}
9093          Maximum number of expressions that can be nested within parentheses by the expr utility.
9094          Value: 32

9095      {_POSIX2_LINE_MAX}
9096          Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
9097          standard input or another file), when the utility is described as processing text files. The
9098          length includes room for the trailing <newline>.
9099          Value: 2 048

9100      {_POSIX2_RE_DUP_MAX}
9101          Maximum number of repeated occurrences of a regular expression permitted when using
9102          the interval notation \{m,n\}; see Chapter 9 (on page 169).
9103          Value: 255

9104  XSI      {_XOPEN_IOV_MAX}
9105          Maximum number of iovec structures that one process has available for use with readv() or
9106          writev().
9107          Value: 16

9108  XSI      {_XOPEN_NAME_MAX}
9109          Maximum number of bytes in a filename (not including the terminating null).
9110          Value: 255

9111  XSI      {_XOPEN_PATH_MAX}
9112          Maximum number of bytes in a pathname.
9113          Value: 1 024

9114      Numerical Limits

9115      The values in the following lists shall be defined in <limits.h> and are constant expressions
9116  XSI      suitable for use in #if preprocessing directives. Moreover, except for {CHAR_BIT}, {DBL_DIG},
9117          {DBL_MAX}, {FLT_DIG}, {FLT_MAX}, {LONG_BIT}, {WORD_BIT}, and {MB_LEN_MAX}, the
9118          symbolic names are defined as expressions of the correct type.

9119      If the value of an object of type char is treated as a signed integer when used in an expression,
9120          the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}
9121          is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value
9122          of {CHAR_MAX} is the same as that of {UCHAR_MAX}.

```


9123		{CHAR_BIT}
9124		Number of bits in a type char .
9125	CX	Value: 8
9126		{CHAR_MAX}
9127		Maximum value of type char .
9128		Value: {UCHAR_MAX} or {SCHAR_MAX}
9129		{CHAR_MIN}
9130		Minimum value of type char .
9131		Value: {SCHAR_MIN} or 0
9132		{INT_MAX}
9133		Maximum value of an int .
9134		Minimum Acceptable Value: 2 147 483 647
9135	XSI	{LONG_BIT}
9136		Number of bits in a long .
9137		Minimum Acceptable Value: 32
9138		{LONG_MAX}
9139		Maximum value of a long .
9140		Minimum Acceptable Value: +2 147 483 647
9141		{MB_LEN_MAX}
9142		Maximum number of bytes in a character, for any supported locale.
9143		Minimum Acceptable Value: 1
9144		{SCHAR_MAX}
9145		Maximum value of type signed char .
9146	CX	Value: +127
9147		{SHRT_MAX}
9148		Maximum value of type short .
9149		Minimum Acceptable Value: +32 767
9150		{SSIZE_MAX}
9151		Maximum value of an object of type ssize_t .
9152		Minimum Acceptable Value: {_POSIX_SSIZE_MAX}
9153		{UCHAR_MAX}
9154		Maximum value of type unsigned char .
9155	CX	Value: 255
9156		{UINT_MAX}
9157		Maximum value of type unsigned .
9158		Minimum Acceptable Value: 4 294 967 295
9159		{ULONG_MAX}
9160		Maximum value of type unsigned long .
9161		Minimum Acceptable Value: 4 294 967 295
9162		{USHRT_MAX}
9163		Maximum value for a type unsigned short .
9164		Minimum Acceptable Value: 65 535
9165	XSI	{WORD_BIT}
9166		Number of bits in a word or type int .
9167		Minimum Acceptable Value: 16

9168 {INT_MIN}
 9169 Minimum value of type **int**.
 9170 Maximum Acceptable Value: -2 147 483 647

9171 {LONG_MIN}
 9172 Minimum value of type **long**.
 9173 Maximum Acceptable Value: -2 147 483 647

9174 {SCHAR_MIN}
 9175 Minimum value of type **signed char**.
 9176 CX Value: -128

9177 {SHRT_MIN}
 9178 Minimum value of type **short**.
 9179 Maximum Acceptable Value: -32 767

9180 {LLONG_MIN}
 9181 Minimum value of type **long long**.
 9182 Maximum Acceptable Value: -9 223 372 036 854 775 807

9183 {LLONG_MAX}
 9184 Maximum value of type **long long**.
 9185 Minimum Acceptable Value: +9 223 372 036 854 775 807

9186 {ULLONG_MAX}
 9187 Maximum value of type **unsigned long long**.
 9188 Minimum Acceptable Value: 18 446 744 073 709 551 615

9189 Other Invariant Values

9190 XSI The following constants shall be defined on all implementations in <limits.h>:

9191 XSI {CHARCLASS_NAME_MAX}
 9192 Maximum number of bytes in a character class name.
 9193 Minimum Acceptable Value: 14

9194 XSI {NL_ARGMAX}
 9195 Maximum value of *digit* in calls to the *printf()* and *scanf()* functions.
 9196 Minimum Acceptable Value: 9

9197 XSI {NL_LANGMAX}
 9198 Maximum number of bytes in a *LANG* name.
 9199 Minimum Acceptable Value: 14

9200 XSI {NL_MSGMAX}
 9201 Maximum message number.
 9202 Minimum Acceptable Value: 32 767

9203 XSI {NL_NMAX}
 9204 Maximum number of bytes in an N-to-1 collation mapping.
 9205 Minimum Acceptable Value: No guaranteed value across all conforming implementations.

9206 XSI {NL_SETMAX}
 9207 Maximum set number.
 9208 Minimum Acceptable Value: 255

9209 XSI {NL_TEXTMAX}
 9210 Maximum number of bytes in a message string.
 9211 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

9212 XSI {NZERO}
 9213 Default process priority.
 9214 Minimum Acceptable Value: 20

9215 APPLICATION USAGE

9216 None.

9217 RATIONALE

9218 A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified
 9219 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request
 9220 for several reasons:

- 9221 • They wanted to avoid making any changes to the standard that could break conforming
 9222 applications, and the requested change could have that effect.
- 9223 • The use of multiple hard links to a file cannot always be replaced with use of symbolic links.
 9224 Symbolic links are semantically different from hard links in that they associate a pathname
 9225 with another pathname rather than a pathname with a file. This has implications for access
 9226 control, file permanence, and transparency.
- 9227 • The original standard developers had considered the issue of allowing for implementations
 9228 that did not in general support hard links, and decided that this would reduce consensus on
 9229 the standard.

9230 Systems that support historical versions of the development option of the ISO POSIX-2 standard
 9231 retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

9232 {PATH_MAX}
 9233 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the
 9234 definition of pathname and the description of {PATH_MAX}, allowing application writers to
 9235 allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has been
 9236 removed by correction to the {PATH_MAX} definition to include the null character. With
 9237 this change, applications that previously allocated {PATH_MAX} bytes will continue to
 9238 succeed.

9239 {SYMLINK_MAX}
 9240 This symbol refers to space for data that is stored in the file system, as opposed to
 9241 {PATH_MAX} which is the length of a name that can be passed to a function. In some
 9242 existing implementations, the filenames pointed to by symbolic links are stored in the
 9243 inodes of the links, so it is important that {SYMLINK_MAX} not be constrained to be as
 9244 large as {PATH_MAX}.

9245 FUTURE DIRECTIONS

9246 None.

9247 SEE ALSO

9248 The System Interfaces volume of IEEE Std 1003.1-2001, *fpathconf()*, *pathconf()*, *sysconf()*

9249 CHANGE HISTORY

9250 First released in Issue 1.

9251 Issue 5

9252 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 9253 Threads Extension.

9254 {FILESIZEBITS} is added for the Large File Summit extensions.

9255 The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to
 9256 make 32-bit values the minimum requirement.

9257 The entry is restructured to improve readability.

9258 **Issue 6**

9259 The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR_MIN},
 9260 {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are maximum
 9261 acceptable values.

9262 The following new requirements on POSIX implementations derive from alignment with the
 9263 Single UNIX Specification:

- 9264 • The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.
- 9265 • The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.
- 9266 • The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

9267 Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX},
 9268 {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

9269 The following values are added for alignment with IEEE Std 1003.1d-1999:

9270 {_POSIX_SS_REPL_MAX}
 9271 {SS_REPL_MAX}
 9272 {POSIX_ALLOC_SIZE_MIN}
 9273 {POSIX_REC_INCR_XFER_SIZE}
 9274 {POSIX_REC_MAX_XFER_SIZE}
 9275 {POSIX_REC_MIN_XFER_SIZE}
 9276 {POSIX_REC_XFER_ALIGN}

9277 Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN}
 9278 for alignment with IEEE Std 1003.1j-2000.

9279 The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment
 9280 with the ISO / IEC 9899: 1999 standard.

9281 The following values are added for alignment with IEEE Std 1003.1q-2000:

9282 {_POSIX_TRACE_EVENT_NAME_MAX}
 9283 {_POSIX_TRACE_NAME_MAX}
 9284 {_POSIX_TRACE_SYS_MAX}
 9285 {_POSIX_TRACE_USER_EVENT_MAX}
 9286 {TRACE_EVENT_NAME_MAX}
 9287 {TRACE_NAME_MAX}
 9288 {TRACE_SYS_MAX}
 9289 {TRACE_USER_EVENT_MAX}

9290 The new limits {_XOPEN_NAME_MAX} and {_XOPEN_PATH_MAX} are added as minimum
 9291 values for {PATH_MAX} and {NAME_MAX} limits on XSI-conformant systems.

9292 The legacy symbols {PASS_MAX} and {TMP_MAX} are removed.

9293 The values for the limits {CHAR_BIT}, {SCHAR_MAX}, and {UCHAR_MAX} are now required
 9294 to be 8, +127, and 255, respectively.

9295 The value for the limit {CHAR_MAX} is now {UCHAR_MAX} or {SCHAR_MAX}.

9296 The value for the limit {CHAR_MIN} is now {SCHAR_MIN} or zero.

9297 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/10 is applied, correcting the value of
 9298 {_POSIX_CHILD_MAX} from 6 to 25. This is for FIPS 151-2 alignment. |

9299 **NAME**

9300 locale.h — category macros

9301 **SYNOPSIS**

9302 #include <locale.h>

9303 **DESCRIPTION**

9304 CX Some of the functionality described on this reference page extends the ISO C standard. Any
9305 conflict between the requirements described here and the ISO C standard is unintentional. This
9306 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

9307 The <locale.h> header shall provide a definition for **lconv** structure, which shall include at least
9308 the following members. (See the definitions of *LC_MONETARY* in Section 7.3.3 (on page 142)
9309 and Section 7.3.4 (on page 145).)

```
9310 char    *currency_symbol
9311 char    *decimal_point
9312 char    frac_digits
9313 char    *grouping
9314 char    *int_curr_symbol
9315 char    int_frac_digits
9316 char    int_n_cs_precedes
9317 char    int_n_sep_by_space
9318 char    int_n_sign_posn
9319 char    int_p_cs_precedes
9320 char    int_p_sep_by_space
9321 char    int_p_sign_posn
9322 char    *mon_decimal_point
9323 char    *mon_grouping
9324 char    *mon_thousands_sep
9325 char    *negative_sign
9326 char    n_cs_precedes
9327 char    n_sep_by_space
9328 char    n_sign_posn
9329 char    *positive_sign
9330 char    p_cs_precedes
9331 char    p_sep_by_space
9332 char    p_sign_posn
9333 char    *thousands_sep
```

9334 The <locale.h> header shall define NULL (as defined in <stddef.h>) and at least the following as
9335 macros:

```
9336 LC_ALL
9337 LC_COLLATE
9338 LC_CTYPE
9339 CX LC_MESSAGES
9340 LC_MONETARY
9341 LC_NUMERIC
9342 LC_TIME
```

9343 which shall expand to distinct integer constant expressions, for use as the first argument to the
9344 *setlocale()* function.

9345 Additional macro definitions, beginning with the characters *LC_* and an uppercase letter, may
9346 also be given here.

9347 The following shall be declared as functions and may also be defined as macros. Function
9348 prototypes shall be provided.

9349 struct lconv *localeconv (void);
9350 char *setlocale(int, const char *);

9351 **APPLICATION USAGE**

9352 None.

9353 **RATIONALE**

9354 None.

9355 **FUTURE DIRECTIONS**

9356 None.

9357 **SEE ALSO**

9358 The System Interfaces volume of IEEE Std 1003.1-2001, *localeconv()*, *setlocale()*, Chapter 8 (on
9359 page 161)

9360 **CHANGE HISTORY**

9361 First released in Issue 3.

9362 Included for alignment with the ISO C standard.

9363 **Issue 6**

9364 The **lconv** structure is expanded with new members (**int_n_cs_precedes**, **int_n_sep_by_space**,
9365 **int_n_sign_posn**, **int_p_cs_precedes**, **int_p_sep_by_space**, and **int_p_sign_posn**) for alignment
9366 with the ISO / IEC 9899:1999 standard.

9367 Extensions beyond the ISO C standard are marked.

9368 **NAME**

9369 math.h — mathematical declarations

9370 **SYNOPSIS**

9371 #include <math.h>

9372 **DESCRIPTION**

9373 CX Some of the functionality described on this reference page extends the ISO C standard.
 9374 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 9375 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 9376 symbols in this header.

9377 The <math.h> header shall include definitions for at least the following types:

9378 **float_t** A real-floating type at least as wide as **float**.

9379 **double_t** A real-floating type at least as wide as **double**, and at least as wide as **float_t**.

9380 If FLT_EVAL_METHOD equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if
 9381 FLT_EVAL_METHOD equals 1, they shall both be **double**; if FLT_EVAL_METHOD equals 2,
 9382 they shall both be **long double**; for other values of FLT_EVAL_METHOD, they are otherwise
 9383 implementation-defined.

9384 The <math.h> header shall define the following macros, where real-floating indicates that the
 9385 argument shall be an expression of real-floating type:

```
9386 int fpclassify(real-floating x);
9387 int isfinite(real-floating x);
9388 int isinf(real-floating x);
9389 int isnan(real-floating x);
9390 int isnormal(real-floating x);
9391 int signbit(real-floating x);
9392 int isgreater(real-floating x, real-floating y);
9393 int isgreaterequal(real-floating x, real-floating y);
9394 int isless(real-floating x, real-floating y);
9395 int islessequal(real-floating x, real-floating y);
9396 int islessgreater(real-floating x, real-floating y);
9397 int unordered(real-floating x, real-floating y);
```

9398 The <math.h> header shall provide for the following constants. The values are of type **double**
 9399 and are accurate within the precision of the **double** type.

9400	XSI	<u>M_E</u>	Value of e
9401		<u>M_LOG2E</u>	Value of $\log_2 e$
9402		<u>M_LOG10E</u>	Value of $\log_{10} e$
9403		<u>M_LN2</u>	Value of $\log_e 2$
9404		<u>M_LN10</u>	Value of $\log_e 10$
9405		<u>M_PI</u>	Value of π
9406		<u>M_PI_2</u>	Value of $\pi/2$
9407		<u>M_PI_4</u>	Value of $\pi/4$
9408		<u>M_1_PI</u>	Value of $1/\pi$
9409		<u>M_2_PI</u>	Value of $2/\pi$

9410	M_2_SQRTPI	Value of $2/\sqrt{\pi}$
9411	M_SQRT2	Value of $\sqrt{2}$
9412	M_SQRT1_2	Value of $1/\sqrt{2}$
9413	The header shall define the following symbolic constants:	
9414	MAXFLOAT	Value of maximum non-infinite single-precision floating-point number.
9415	HUGE_VAL	A positive double expression, not necessarily representable as a float . Used
9416		as an error value returned by the mathematics library. HUGE_VAL evaluates
9417		to +infinity on systems supporting IEEE Std 754-1985.
9418	HUGE_VALF	A positive float constant expression. Used as an error value returned by the
9419		mathematics library. HUGE_VALF evaluates to +infinity on systems
9420		supporting IEEE Std 754-1985.
9421	HUGE_VALL	A positive long double constant expression. Used as an error value returned
9422		by the mathematics library. HUGE_VALL evaluates to +infinity on systems
9423		supporting IEEE Std 754-1985.
9424	INFINITY	A constant expression of type float representing positive or unsigned infinity,
9425		if available; else a positive constant of type float that overflows at translation
9426		time.
9427	NAN	A constant expression of type float representing a quiet NaN. This symbolic
9428		constant is only defined if the implementation supports quiet NaNs for the
9429		float type.
9430	The following macros shall be defined for number classification. They represent the mutually-	
9431	exclusive kinds of floating-point values. They expand to integer constant expressions with	
9432	distinct values. Additional implementation-defined floating-point classifications, with macro	
9433	definitions beginning with FP_ and an uppercase letter, may also be specified by the	
9434	implementation.	
9435	FP_INFINITE	
9436	FP_NAN	
9437	FP_NORMAL	
9438	FP_SUBNORMAL	
9439	FP_ZERO	
9440	The following optional macros indicate whether the <i>fma()</i> family of functions are fast compared	
9441	with direct code:	
9442	FP_FAST_FMA	
9443	FP_FAST_FMAF	
9444	FP_FAST_FMAL	
9445	The FP_FAST_FMA macro shall be defined to indicate that the <i>fma()</i> function generally executes	
9446	about as fast as, or faster than, a multiply and an add of double operands. The other macros	
9447	have the equivalent meaning for the float and long double versions.	
9448	The following macros shall expand to integer constant expressions whose values are returned by	
9449	<i>ilogb(x)</i> if <i>x</i> is zero or NaN, respectively. The value of FP_ILOGB0 shall be either {INT_MIN} or	
9450	-{INT_MAX}. The value of FP_ILOGBNAN shall be either {INT_MAX} or {INT_MIN}.	
9451	FP_ILOGB0	
9452	FP_ILOGBNAN	

9453 The following macros shall expand to the integer constants 1 and 2, respectively;

9454 **MATH_ERRNO**

9455 **MATH_ERREXCEPT**

9456 The following macro shall expand to an expression that has type **int** and the value

9457 **MATH_ERRNO**, **MATH_ERREXCEPT**, or the bitwise-inclusive OR of both:

9458 **math_errhandling**

9459 The value of **math_errhandling** is constant for the duration of the program. It is unspecified

9460 whether **math_errhandling** is a macro or an identifier with external linkage. If a macro definition

9461 is suppressed or a program defines an identifier with the name **math_errhandling**, the behavior

9462 is undefined. If the expression (**math_errhandling** & **MATH_ERREXCEPT**) can be non-zero, the

9463 implementation shall define the macros **FE_DIVBYZERO**, **FE_INVALID**, and **FE_OVERFLOW** in

9464 **<fenv.h>**.

9465 The following shall be declared as functions and may also be defined as macros. Function

9466 prototypes shall be provided.

9467 double **acos**(double);

9468 float **acosf**(float);

9469 double **acosh**(double);

9470 float **acoshf**(float);

9471 long double **acoshl**(long double);

9472 long double **acosl**(long double);

9473 double **asin**(double);

9474 float **asinf**(float);

9475 double **asinh**(double);

9476 float **asinhf**(float);

9477 long double **asinh1**(long double);

9478 long double **asinl**(long double);

9479 double **atan**(double);

9480 double **atan2**(double, double);

9481 float **atan2f**(float, float);

9482 long double **atan2l**(long double, long double);

9483 float **atanf**(float);

9484 double **atanh**(double);

9485 float **atanhf**(float);

9486 long double **atanh1**(long double);

9487 long double **atanl**(long double);

9488 double **cbrt**(double);

9489 float **cbrtf**(float);

9490 long double **cbrtl**(long double);

9491 double **ceil**(double);

9492 float **ceilf**(float);

9493 long double **ceil1**(long double);

9494 double **copysign**(double, double);

9495 float **copysignf**(float, float);

9496 long double **copysignl**(long double, long double);

9497 double **cos**(double);

9498 float **cosf**(float);

9499 double **cosh**(double);

9500 float **coshf**(float);

9501 long double **cosh1**(long double);

```

9502     long double cosl(long double);
9503     double      erf(double);
9504     double      erfc(double);
9505     float       erfcf(float);
9506     long double erfc1(long double);
9507     float       erff(float);
9508     long double erfl(long double);
9509     double      exp(double);
9510     double      exp2(double);
9511     float       exp2f(float);
9512     long double exp2l(long double);
9513     float       expf(float);
9514     long double expl(long double);
9515     double      expm1(double);
9516     float       expm1f(float);
9517     long double expm1l(long double);
9518     double      fabs(double);
9519     float       fabsf(float);
9520     long double fabs1(long double);
9521     double      fdim(double, double);
9522     float       fdimf(float, float);
9523     long double fdiml(long double, long double);
9524     double      floor(double);
9525     float       floorf(float);
9526     long double floorl(long double);
9527     double      fma(double, double, double);
9528     float       fmaf(float, float, float);
9529     long double fmal(long double, long double, long double);
9530     double      fmax(double, double);
9531     float       fmaxf(float, float);
9532     long double fmaxl(long double, long double);
9533     double      fmin(double, double);
9534     float       fminf(float, float);
9535     long double fminl(long double, long double);
9536     double      fmod(double, double);
9537     float       fmodf(float, float);
9538     long double fmodl(long double, long double);
9539     double      frexp(double, int *);
9540     float       frexpf(float value, int *);
9541     long double frexpl(long double value, int *);
9542     double      hypot(double, double);
9543     float       hypotf(float, float);
9544     long double hypotl(long double, long double);
9545     int         ilogb(double);
9546     int         ilogbf(float);
9547     int         ilogbl(long double);
9548 xsi  double     j0(double);
9549     double     j1(double);
9550     double     jn(int, double);
9551     double     ldexp(double, int);
9552     float     ldexpf(float, int);
9553     long double ldexpl(long double, int);

```

```

9554     double      lgamma(double);
9555     float        lgammaf(float);
9556     long double  lgammal(long double);
9557     long long    llrint(double);
9558     long long    llrintf(float);
9559     long long    llrintl(long double);
9560     long long    llround(double);
9561     long long    llroundf(float);
9562     long long    llroundl(long double);
9563     double       log(double);
9564     double       log10(double);
9565     float        log10f(float);
9566     long double  log10l(long double);
9567     double       log1p(double);
9568     float        log1pf(float);
9569     long double  log1pl(long double);
9570     double       log2(double);
9571     float        log2f(float);
9572     long double  log2l(long double);
9573     double       logb(double);
9574     float        logbf(float);
9575     long double  logbl(long double);
9576     float        logf(float);
9577     long double  logl(long double);
9578     long         lrint(double);
9579     long         lrintf(float);
9580     long         lrintl(long double);
9581     long         lround(double);
9582     long         lroundf(float);
9583     long         lroundl(long double);
9584     double       modf(double, double *);
9585     float        modff(float, float *);
9586     long double  modfl(long double, long double *);
9587     double       nan(const char *);
9588     float        nanf(const char *);
9589     long double  nanl(const char *);
9590     double       nearbyint(double);
9591     float        nearbyintf(float);
9592     long double  nearbyintl(long double);
9593     double       nextafter(double, double);
9594     float        nextafterf(float, float);
9595     long double  nextafterl(long double, long double);
9596     double       nexttoward(double, long double);
9597     float        nexttowardf(float, long double);
9598     long double  nexttowardl(long double, long double);
9599     double       pow(double, double);
9600     float        powf(float, float);
9601     long double  powl(long double, long double);
9602     double       remainder(double, double);
9603     float        remainderf(float, float);
9604     long double  remainderl(long double, long double);
9605     double       remquo(double, double, int *);

```

```

9606 float      remquof(float, float, int *);
9607 long double remquol(long double, long double, int *);
9608 double      rint(double);
9609 float      rintf(float);
9610 long double rintl(long double);
9611 double      round(double);
9612 float      roundf(float);
9613 long double roundl(long double);
9614 XSI double   scalb(double, double);
9615 double      scalbln(double, long);
9616 float      scalblnf(float, long);
9617 long double scalblnl(long double, long);
9618 double      scalbn(double, int);
9619 float      scalbnf(float, int);
9620 long double scalbnl(long double, int);
9621 double      sin(double);
9622 float      sinf(float);
9623 double      sinh(double);
9624 float      sinhf(float);
9625 long double sinhl(long double);
9626 long double sinl(long double);
9627 double      sqrt(double);
9628 float      sqrtf(float);
9629 long double sqrtl(long double);
9630 double      tan(double);
9631 float      tanf(float);
9632 double      tanh(double);
9633 float      tanhf(float);
9634 long double tanhl(long double);
9635 long double tanl(long double);
9636 double      tgamma(double);
9637 float      tgammaf(float);
9638 long double tgamma1(long double);
9639 double      trunc(double);
9640 float      truncf(float);
9641 long double trunc1(long double);
9642 XSI double   y0(double);
9643 double      y1(double);
9644 double      yn(int, double);
9645

```

9646 The following external variable shall be defined:

```

9647 XSI extern int signgam;
9648

```

9649 The behavior of each of the functions defined in <math.h> is specified in the System Interfaces
9650 volume of IEEE Std 1003.1-2001 for all representable values of its input arguments, except where
9651 stated otherwise. Each function shall execute as if it were a single operation without generating
9652 any externally visible exceptional conditions.

9653 APPLICATION USAGE

9654 The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is
9655 off) the implementation to contract expressions. Each pragma can occur either outside external
9656 declarations or preceding all explicit declarations and statements inside a compound statement.
9657 When outside external declarations, the pragma takes effect from its occurrence until another
9658 FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a
9659 compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT
9660 pragma is encountered (including within a nested compound statement), or until the end of the
9661 compound statement; at the end of a compound statement the state for the pragma is restored to
9662 its condition just before the compound statement. If this pragma is used in any other context, the
9663 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

9664 RATIONALE

9665 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type
9666 **double**. All the names formed by appending 'f' or 'l' to a name in <math.h> were reserved
9667 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999
9668 standard provides for all three versions of math functions.

9669 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their
9670 capability is available through *sprintf()*. These are provided on XSI-conformant systems
9671 supporting the Legacy Option Group.

9672 FUTURE DIRECTIONS

9673 None.

9674 SEE ALSO

9675 <stddef.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *acos()*, *acosh()*,
9676 *asin()*, *atan()*, *atan2()*, *cbrt()*, *ceil()*, *cos()*, *cosh()*, *erf()*, *exp()*, *expm1()*, *fabs()*, *floor()*, *fmod()*,
9677 *frexp()*, *hypot()*, *ilogb()*, *isnan()*, *j0()*, *ldexp()*, *lgamma()*, *log()*, *log10()*, *log1p()*, *logb()*, *modf()*,
9678 *nextafter()*, *pow()*, *remainder()*, *rint()*, *scalb()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *y0()*

9679 CHANGE HISTORY

9680 First released in Issue 1.

9681 Issue 6

9682 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

9683 **NAME**

9684 monetary.h — monetary types

9685 **SYNOPSIS**

9686 xSI #include <monetary.h>

9687

9688 **DESCRIPTION**9689 The **<monetary.h>** header shall define the following types:9690 **size_t** As described in **<stddef.h>**.9691 **ssize_t** As described in **<sys/types.h>**.9692 The following shall be declared as a function and may also be defined as a macro. A function
9693 prototype shall be provided.9694 **ssize_t** strfmon(char *restrict, size_t, const char *restrict, ...);9695 **APPLICATION USAGE**

9696 None.

9697 **RATIONALE**

9698 None.

9699 **FUTURE DIRECTIONS**

9700 None.

9701 **SEE ALSO**9702 The System Interfaces volume of IEEE Std 1003.1-2001, *strfmon()*9703 **CHANGE HISTORY**

9704 First released in Issue 4.

9705 **Issue 6**9706 The **restrict** keyword is added to the prototype for *strfmon()*.

9707 **NAME**

9708 mqqueue.h — message queues (**REALTIME**)

9709 **SYNOPSIS**

9710 MSG #include <mqqueue.h>

9711

9712 **DESCRIPTION**

9713 The <mqqueue.h> header shall define the **mqd_t** type, which is used for message queue
9714 descriptors. This is not an array type.

9715 The <mqqueue.h> header shall define the **sigevent** structure (as described in <signal.h>) and the
9716 **mq_attr** structure, which is used in getting and setting the attributes of a message queue.
9717 Attributes are initially set when the message queue is created. An **mq_attr** structure shall have at
9718 least the following fields:

9719	long	mq_flags	Message queue flags.
9720	long	mq_maxmsg	Maximum number of messages.
9721	long	mq_msgsize	Maximum message size.
9722	long	mq_curmsgs	Number of messages currently queued.

9723 The following shall be declared as functions and may also be defined as macros. Function
9724 prototypes shall be provided.

9725	int	mq_close(mqd_t);
9726	int	mq_getattr(mqd_t, struct mq_attr *);
9727	int	mq_notify(mqd_t, const struct sigevent *);
9728	mqd_t	mq_open(const char *, int, ...);
9729	ssize_t	mq_receive(mqd_t, char *, size_t, unsigned *);
9730	int	mq_send(mqd_t, const char *, size_t, unsigned);
9731	int	mq_setattr(mqd_t, const struct mq_attr *restrict,
9732		struct mq_attr *restrict);
9733	TMO	ssize_t mq_timedreceive(mqd_t, char *restrict, size_t,
9734		unsigned *restrict, const struct timespec *restrict);
9735	int	mq_timedsend(mqd_t, const char *, size_t, unsigned,
9736		const struct timespec *);
9737	int	mq_unlink(const char *);

9738 Inclusion of the <mqqueue.h> header may make visible symbols defined in the headers <fcntl.h>,
9739 <signal.h>, <sys/types.h>, and <time.h>.

9740 **APPLICATION USAGE**

9741 None.

9742 **RATIONALE**

9743 None.

9744 **FUTURE DIRECTIONS**

9745 None.

9746 **SEE ALSO**

9747 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
9748 IEEE Std 1003.1-2001, *mq_close()*, *mq_getattr()*, *mq_notify()*, *mq_open()*, *mq_receive()*, *mq_send()*,
9749 *mq_setattr()*, *mq_timedreceive()*, *mq_timedsend()*, *mq_unlink()*

9750 CHANGE HISTORY

9751 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

9752 Issue 6

9753 The **<mqueue.h>** header is marked as part of the Message Passing option.

9754 The *mq_timedreceive()* and *mq_timedsend()* functions are added for alignment with
9755 IEEE Std 1003.1d-1999.

9756 The **restrict** keyword is added to the prototypes for *mq_setattr()* and *mq_timedreceive()*.

9757 **NAME**

9758 ndbm.h — definitions for ndbm database operations

9759 **SYNOPSIS**9760 xSI `#include <ndbm.h>`

9761

9762 **DESCRIPTION**9763 The <ndbm.h> header shall define the **datum** type as a structure that includes at least the
9764 following members:9765 `void *dptr` A pointer to the application's data.9766 `size_t dsize` The size of the object pointed to by *dptr*.9767 The **size_t** type shall be defined as described in <stddef.h>.9768 The <ndbm.h> header shall define the **DBM** type.9769 The following constants shall be defined as possible values for the *store_mode* argument to
9770 *dbm_store()*:9771 `DBM_INSERT` Insertion of new entries only.9772 `DBM_REPLACE` Allow replacing existing entries.9773 The following shall be declared as functions and may also be defined as macros. Function
9774 prototypes shall be provided.9775 `int dbm_clearerr(DBM *);`9776 `void dbm_close(DBM *);`9777 `int dbm_delete(DBM *, datum);`9778 `int dbm_error(DBM *);`9779 `datum dbm_fetch(DBM *, datum);`9780 `datum dbm_firstkey(DBM *);`9781 `datum dbm_nextkey(DBM *);`9782 `DBM *dbm_open(const char *, int, mode_t);`9783 `int dbm_store(DBM *, datum, datum, int);`9784 The **mode_t** type shall be defined through **typedef** as described in <sys/types.h>.9785 **APPLICATION USAGE**

9786 None.

9787 **RATIONALE**

9788 None.

9789 **FUTURE DIRECTIONS**

9790 None.

9791 **SEE ALSO**9792 <stddef.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *dbm_clearerr()*9793 **CHANGE HISTORY**

9794 First released in Issue 4, Version 2.

9795 **Issue 5**9796 References to the definitions of **size_t** and **mode_t** are added to the DESCRIPTION.

9797 **NAME**

9798 net/if.h — sockets local interfaces

9799 **SYNOPSIS**

9800 #include <net/if.h>

9801 **DESCRIPTION**9802 The **<net/if.h>** header shall define the **if_nameindex** structure that includes at least the
9803 following members:

9804 unsigned if_index Numeric index of the interface.

9805 char *if_name Null-terminated name of the interface.

9806 The **<net/if.h>** header shall define the following macro for the length of a buffer containing an
9807 interface name (including the terminating NULL character):

9808 IF_NAMESIZE Interface name length.

9809 The following shall be declared as functions and may also be defined as macros. Function
9810 prototypes shall be provided.

9811 unsigned if_nametoindex(const char *);

9812 char *if_indextoname(unsigned, char *);

9813 struct if_nameindex *if_nameindex(void);

9814 void if_freenameindex(struct if_nameindex *);

9815 **APPLICATION USAGE**

9816 None.

9817 **RATIONALE**

9818 None.

9819 **FUTURE DIRECTIONS**

9820 None.

9821 **SEE ALSO**9822 The System Interfaces volume of IEEE Std 1003.1-2001, *if_freenameindex()*, *if_indextoname()*,
9823 *if_nameindex()*, *if_nametoindex()*9824 **CHANGE HISTORY**

9825 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9826 **NAME**

9827 netdb.h — definitions for network database operations

9828 **SYNOPSIS**

9829 #include <netdb.h>

9830 **DESCRIPTION**

9831 The <netdb.h> header may define the **in_port_t** type and the **in_addr_t** type as described in
9832 <netinet/in.h>.

9833 The <netdb.h> header shall define the **hostent** structure that includes at least the following
9834 members:

9835	char	*h_name	Official name of the host.
9836	char	**h_aliases	A pointer to an array of pointers to
9837			alternative host names, terminated by a
9838			null pointer.
9839	int	h_addrtype	Address type.
9840	int	h_length	The length, in bytes, of the address.
9841	char	**h_addr_list	A pointer to an array of pointers to network
9842			addresses (in network byte order) for the host,
9843			terminated by a null pointer.

9844 The <netdb.h> header shall define the **netent** structure that includes at least the following
9845 members:

9846	char	*n_name	Official, fully-qualified (including the
9847			domain) name of the host.
9848	char	**n_aliases	A pointer to an array of pointers to
9849			alternative network names, terminated by a
9850			null pointer.
9851	int	n_addrtype	The address type of the network.
9852	uint32_t	n_net	The network number, in host byte order.

9853 The **uint32_t** type shall be defined as described in <inttypes.h>.

9854 The <netdb.h> header shall define the **protoent** structure that includes at least the following
9855 members:

9856	char	*p_name	Official name of the protocol.
9857	char	**p_aliases	A pointer to an array of pointers to
9858			alternative protocol names, terminated by
9859			a null pointer.
9860	int	p_proto	The protocol number.

9861 The <netdb.h> header shall define the **servent** structure that includes at least the following
9862 members:

9863	char	*s_name	Official name of the service.
9864	char	**s_aliases	A pointer to an array of pointers to
9865			alternative service names, terminated by
9866			a null pointer.
9867	int	s_port	The port number at which the service
9868			resides, in network byte order.
9869	char	*s_proto	The name of the protocol to use when
9870			contacting the service.

9871 The <netdb.h> header shall define the IPPORT_RESERVED macro with the value of the highest
 9872 reserved Internet port number.

9873 OB When the <netdb.h> header is included, *h_errno* shall be available as a modifiable lvalue of type
 9874 **int**. It is unspecified whether *h_errno* is a macro or an identifier declared with external linkage.

9875 The <netdb.h> header shall define the following macros for use as error values for
 9876 *gethostbyaddr()* and *gethostbyname()*:

9877 HOST_NOT_FOUND
 9878 NO_DATA
 9879 NO_RECOVERY
 9880 TRY_AGAIN

9881 Address Information Structure

9882 The <netdb.h> header shall define the **addrinfo** structure that includes at least the following
 9883 members:

9884	int	ai_flags	Input flags.
9885	int	ai_family	Address family of socket.
9886	int	ai_socktype	Socket type.
9887	int	ai_protocol	Protocol of socket.
9888	socklen_t	ai_addrlen	Length of socket address.
9889	struct sockaddr	*ai_addr	Socket address of socket.
9890	char	*ai_canonname	Canonical name of service location.
9891	struct addrinfo	*ai_next	Pointer to next in list.

9892 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 9893 constants for use in the *flags* field of the **addrinfo** structure:

9894 AI_PASSIVE Socket address is intended for *bind()*.

9895 AI_CANONNAME
 9896 Request for canonical name.

9897 AI_NUMERICHOST
 9898 Return numeric host address as name.

9899 AI_NUMERICSERV
 9900 Inhibit service name resolution.

9901 AI_V4MAPPED If no IPv6 addresses are found, query for IPv4 addresses and return them to
 9902 the caller as IPv4-mapped IPv6 addresses.

9903 AI_ALL Query for both IPv4 and IPv6 addresses.

9904 AI_ADDRCONFIG
 9905 Query for IPv4 addresses only when an IPv4 address is configured; query for
 9906 IPv6 addresses only when an IPv6 address is configured.

9907 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 9908 constants for use in the *flags* argument to *getnameinfo()*:

9909 NI_NOFQDN Only the nodename portion of the FQDN is returned for local hosts.

9910 NI_NUMERICHOST
 9911 The numeric form of the node's address is returned instead of its name.

```

9912      NI_NAMEREQD Return an error if the node's name cannot be located in the database.
9913      NI_NUMERICSERV
9914          The numeric form of the service address is returned instead of its name.
9915      NI_NUMERICSCOPE
9916          For IPv6 addresses, the numeric form of the scope identifier is returned
9917          instead of its name.
9918      NI_DGRAM      Indicates that the service is a datagram service (SOCK_DGRAM).

9919      Address Information Errors

9920      The <netdb.h> header shall define the following macros for use as error values for getaddrinfo()
9921      and getnameinfo():

9922      EAI_AGAIN      The name could not be resolved at this time. Future attempts may succeed.
9923      EAI_BADFLAGS   The flags had an invalid value.
9924      EAI_FAIL       A non-recoverable error occurred.
9925      EAI_FAMILY     The address family was not recognized or the address length was invalid for
9926      the specified family.
9927      EAI_MEMORY     There was a memory allocation failure.
9928      EAI_NONAME     The name does not resolve for the supplied parameters.
9929                  NI_NAMEREQD is set and the host's name cannot be located, or both
9930                  nodename and servname were null.
9931      EAI_SERVICE    The service passed was not recognized for the specified socket type.
9932      EAI_SOCKETYPE  The intended socket type was not recognized.
9933      EAI_SYSTEM     A system error occurred. The error code can be found in errno.
9934      EAI_OVERFLOW
9935          An argument buffer overflowed.

9936      The following shall be declared as functions and may also be defined as macros. Function
9937      prototypes shall be provided.

9938      void          endhostent(void);
9939      void          endnetent(void);
9940      void          endprotoent(void);
9941      void          endservent(void);
9942      void          freeaddrinfo(struct addrinfo *);
9943      const char    *gai_strerror(int);
9944      int           getaddrinfo(const char *restrict, const char *restrict,
9945                               const struct addrinfo *restrict,
9946                               struct addrinfo **restrict);
9947      struct hostent *gethostbyaddr(const void *, socklen_t, int);
9948      struct hostent *gethostbyname(const char *);
9949      struct hostent *gethostent(void);
9950      int           getnameinfo(const struct sockaddr *restrict, socklen_t,
9951                               char *restrict, socklen_t, char *restrict,
9952                               socklen_t, int);
9953      struct netent *getnetbyaddr(uint32_t, int);
9954      struct netent *getnetbyname(const char *);

```

```

9955      struct netent      *getnetent(void);
9956      struct protoent    *getprotobyname(const char *);
9957      struct protoent    *getprotobyname(int);
9958      struct protoent    *getprotoent(void);
9959      struct servent      *getservbyname(const char *, const char *);
9960      struct servent      *getservbyport(int, const char *);
9961      struct servent      *getservent(void);
9962      void                sethostent(int);
9963      void                setnetent(int);
9964      void                setprotoent(int);
9965      void                setservent(int);

```

9966 The type **socklen_t** shall be defined through **typedef** as described in <sys/socket.h>.

9967 Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>,
 9968 <sys/socket.h>, and <inttypes.h>.

9969 APPLICATION USAGE

9970 None.

9971 RATIONALE

9972 None.

9973 FUTURE DIRECTIONS

9974 None.

9975 SEE ALSO

9976 <netinet/in.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces volume of
 9977 IEEE Std 1003.1-2001, *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *getaddrinfo()*,
 9978 *getnameinfo()*

9979 CHANGE HISTORY

9980 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9981 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for
 9982 *gai_strerror()* from **char *** to **const char ***. This is for coordination with the IPnG Working Group.

9983 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/11 is applied, adding a description of the
 9984 NI_NUMERICSCOPE macro and correcting the *getnameinfo()* function prototype. These changes
 9985 are for alignment with IPv6.

9986 **NAME**

9987 netinet/in.h — Internet address family

9988 **SYNOPSIS**

9989 #include <netinet/in.h>

9990 **DESCRIPTION**

9991 The <netinet/in.h> header shall define the following types:

9992 **in_port_t** Equivalent to the type **uint16_t** as defined in <inttypes.h>.

9993 **in_addr_t** Equivalent to the type **uint32_t** as defined in <inttypes.h>.

9994 The **sa_family_t** type shall be defined as described in <sys/socket.h>.

9995 The **uint8_t** and **uint32_t** type shall be defined as described in <inttypes.h>. Inclusion of the
9996 <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>.

9997 The <netinet/in.h> header shall define the **in_addr** structure that includes at least the following
9998 member:

9999 in_addr_t s_addr

10000 The <netinet/in.h> header shall define the **sockaddr_in** structure that includes at least the
10001 following members (all in network byte order):

10002 sa_family_t sin_family AF_INET.
10003 in_port_t sin_port Port number.
10004 struct in_addr sin_addr IP address.

10005 The **sockaddr_in** structure is used to store addresses for the Internet address family. Values of
10006 this type shall be cast by applications to **struct sockaddr** for use with socket functions.

10007 IP6 The <netinet/in.h> header shall define the **in6_addr** structure that contains at least the following
10008 member:

10009 uint8_t s6_addr[16]

10010 This array is used to contain a 128-bit IPv6 address, stored in network byte order.

10011 The <netinet/in.h> header shall define the **sockaddr_in6** structure that includes at least the
10012 following members (all in network byte order):

10013 sa_family_t sin6_family AF_INET6.
10014 in_port_t sin6_port Port number.
10015 uint32_t sin6_flowinfo IPv6 traffic class and flow information.
10016 struct in6_addr sin6_addr IPv6 address.
10017 uint32_t sin6_scope_id Set of interfaces for a scope.

10018 The **sockaddr_in6** structure shall be set to zero by an application prior to using it, since
10019 implementations are free to have additional, implementation-defined fields in **sockaddr_in6**.

10020 The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the
10021 scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, the application
10022 shall ensure that *sin6_scope_id* is a link index. For a site scope *sin6_addr*, the application shall
10023 ensure that *sin6_scope_id* is a site index. The mapping of *sin6_scope_id* to an interface or set of
10024 interfaces is implementation-defined.

10025 The <netinet/in.h> header shall declare the following external variable:

10026 const struct in6_addr in6addr_any

10027	This variable is initialized by the system to contain the wildcard IPv6 address. The	
10028	<netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be	
10029	constant at compile time and can be used to initialize a variable of type struct in6_addr to the	
10030	IPv6 wildcard address.	
10031	The <netinet/in.h> header shall declare the following external variable:	
10032	const struct in6_addr in6addr_loopback	
10033	This variable is initialized by the system to contain the loopback IPv6 address. The	
10034	<netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be	
10035	constant at compile time and can be used to initialize a variable of type struct in6_addr to the	
10036	IPv6 loopback address.	
10037	The <netinet/in.h> header shall define the ipv6_mreq structure that includes at least the	
10038	following members:	
10039	struct in6_addr	ipv6mr_multiaddr IPv6 multicast address.
10040	unsigned	ipv6mr_interface Interface index.
10041		
10042	The <netinet/in.h> header shall define the following macros for use as values of the <i>level</i>	
10043	argument of <i>getsockopt()</i> and <i>setsockopt()</i> :	
10044	IPPROTO_IP	Internet protocol.
10045 IP6	IPPROTO_IPV6	Internet Protocol Version 6.
10046	IPPROTO_ICMP	Control message protocol.
10047 RS	IPPROTO_RAW	Raw IP Packets Protocol.
10048	IPPROTO_TCP	Transmission control protocol.
10049	IPPROTO_UDP	User datagram protocol.
10050	The <netinet/in.h> header shall define the following macros for use as destination addresses for	
10051	<i>connect()</i> , <i>sendmsg()</i> , and <i>sendto()</i> :	
10052	INADDR_ANY	IPv4 local host address.
10053	INADDR_BROADCAST	IPv4 broadcast address.
10054	The <netinet/in.h> header shall define the following macro to help applications declare buffers	
10055	of the proper size to store IPv4 addresses in string form:	
10056	INET_ADDRSTRLEN	16. Length of the string form for IP.
10057	The <i>htonl()</i> , <i>htons()</i> , <i>ntohl()</i> , and <i>ntohs()</i> functions shall be available as defined in <arpa/inet.h>.	
10058	Inclusion of the <netinet/in.h> header may also make visible all symbols from <arpa/inet.h>.	
10059 IP6	The <netinet/in.h> header shall define the following macro to help applications declare buffers	
10060	of the proper size to store IPv6 addresses in string form:	
10061	INET6_ADDRSTRLEN	46. Length of the string form for IPv6.
10062	The <netinet/in.h> header shall define the following macros, with distinct integer values, for use	
10063	in the <i>option_name</i> argument in the <i>getsockopt()</i> or <i>setsockopt()</i> functions at protocol level	
10064	IPPROTO_IPV6:	
10065	IPV6_JOIN_GROUP	Join a multicast group.

10066	IPV6_LEAVE_GROUP	Quit a multicast group.
10067	IPV6_MULTICAST_HOPS	
10068		Multicast hop limit.
10069	IPV6_MULTICAST_IF	Interface to use for outgoing multicast packets.
10070	IPV6_MULTICAST_LOOP	
10071		Multicast packets are delivered back to the local application.
10072	IPV6_UNICAST_HOPS	Unicast hop limit.
10073	IPV6_V6ONLY	Restrict AF_INET6 socket to IPv6 communications only.
10074	The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses.	
10075	Each macro is of type int and takes a single argument of type const struct in6_addr * :	
10076	IN6_IS_ADDR_UNSPECIFIED	
10077		Unspecified address.
10078	IN6_IS_ADDR_LOOPBACK	
10079		Loopback address.
10080	IN6_IS_ADDR_MULTICAST	
10081		Multicast address.
10082	IN6_IS_ADDR_LINKLOCAL	
10083		Unicast link-local address.
10084	IN6_IS_ADDR_SITELOCAL	
10085		Unicast site-local address.
10086	IN6_IS_ADDR_V4MAPPED	
10087		IPv4 mapped address.
10088	IN6_IS_ADDR_V4COMPAT	
10089		IPv4-compatible address.
10090	IN6_IS_ADDR_MC_NODELOCAL	
10091		Multicast node-local address.
10092	IN6_IS_ADDR_MC_LINKLOCAL	
10093		Multicast link-local address.
10094	IN6_IS_ADDR_MC_SITELOCAL	
10095		Multicast site-local address.
10096	IN6_IS_ADDR_MC_ORGLOCAL	
10097		Multicast organization-local address.
10098	IN6_IS_ADDR_MC_GLOBAL	
10099		Multicast global address.

10100 **APPLICATION USAGE**

10101 None.

10102 **RATIONALE**

10103 None.

10104 **FUTURE DIRECTIONS**

10105 None.

10106 **SEE ALSO**

10107 Section 4.8 (on page 101), <arpa/inet.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces
 10108 volume of IEEE Std 1003.1-2001, *connect()*, *getsockopt()*, *htonl()*, *htons()*, *ntohl()*, *ntohs()*,
 10109 *sendmsg()*, *sendto()*, *setsockopt()*

10110 **CHANGE HISTORY**

10111 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10112 The *sin_zero* member was removed from the **sockaddr_in** structure as per The Open Group Base
 10113 Resolution bwg2001-004.

10114 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/12 is applied, adding **const** qualifiers to
 10115 the *in6addr_any* and *in6addr_loopback* external variables.

10116 **NAME**

10117 netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10118 **SYNOPSIS**

10119 #include <netinet/tcp.h>

10120 **DESCRIPTION**

10121 The <netinet/tcp.h> header shall define the following macro for use as a socket option at the
10122 IPPROTO_TCP level:

10123 TCP_NODELAY Avoid coalescing of small segments.

10124 The macro shall be defined in the header. The implementation need not allow the value of the
10125 option to be set via *setsockopt()* or retrieved via *getsockopt()*.

10126 **APPLICATION USAGE**

10127 None.

10128 **RATIONALE**

10129 None.

10130 **FUTURE DIRECTIONS**

10131 None.

10132 **SEE ALSO**

10133 <sys/socket.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *getsockopt()*, *setsockopt()*

10134 **CHANGE HISTORY**

10135 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10136 **NAME**

10137 nl_types.h — data types

10138 **SYNOPSIS**

10139 xSI #include <nl_types.h>

10140

10141 **DESCRIPTION**

10142 The <nl_types.h> header shall contain definitions of at least the following types:

10143 **nl_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*
 10144 to identify a catalog descriptor.

10145 **nl_item** Used by *nl_langinfo()* to identify items of *langinfo* data. Values of objects
 10146 of type **nl_item** are defined in <langinfo.h>.

10147 The <nl_types.h> header shall contain definitions of at least the following constants:

10148 **NL_SETD** Used by *gencat* when no *\$set* directive is specified in a message text source
 10149 file; see the Internationalization Guide. This constant can be passed as the
 10150 value of *set_id* on subsequent calls to *catgets()* (that is, to retrieve
 10151 messages from the default message set). The value of **NL_SETD** is
 10152 implementation-defined.

10153 **NL_CAT_LOCALE** Value that must be passed as the *oflag* argument to *catopen()* to ensure
 10154 that message catalog selection depends on the *LC_MESSAGES* locale
 10155 category, rather than directly on the *LANG* environment variable.

10156 The following shall be declared as functions and may also be defined as macros. Function
 10157 prototypes shall be provided.

```
10158 int      catclose(nl_catd);
10159 char     *catgets(nl_catd, int, int, const char *);
10160 nl_catd  catopen(const char *, int);
```

10161 **APPLICATION USAGE**

10162 None.

10163 **RATIONALE**

10164 None.

10165 **FUTURE DIRECTIONS**

10166 None.

10167 **SEE ALSO**

10168 <langinfo.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *catclose()*, *catgets()*,
 10169 *catopen()*, *nl_langinfo()*, the Shell and Utilities volume of IEEE Std 1003.1-2001, *gencat*

10170 **CHANGE HISTORY**

10171 First released in Issue 2.

10172 **NAME**

10173 poll.h — definitions for the poll() function

10174 **SYNOPSIS**

10175 xSI #include <poll.h>

10176

10177 **DESCRIPTION**

10178 The <poll.h> header shall define the **pollfd** structure that includes at least the following
10179 members:

10180 int fd The following descriptor being polled.

10181 short events The input event flags (see below).

10182 short revents The output event flags (see below).

10183 The <poll.h> header shall define the following type through **typedef**:

10184 **nfds_t** An unsigned integer type used for the number of file descriptors.

10185 The implementation shall support one or more programming environments in which the width
10186 of **nfds_t** is no greater than the width of type **long**. The names of these programming
10187 environments can be obtained using the *confstr()* function or the *getconf* utility.

10188 The following symbolic constants shall be defined, zero or more of which may be OR'ed together
10189 to form the *events* or *revents* members in the **pollfd** structure:

10190 POLLIN Data other than high-priority data may be read without blocking.

10191 POLLRDNORM Normal data may be read without blocking.

10192 POLLRDBAND Priority data may be read without blocking.

10193 POLLPRI High priority data may be read without blocking.

10194 POLLOUT Normal data may be written without blocking.

10195 POLLWRNORM Equivalent to POLLOUT.

10196 POLLWRBAND Priority data may be written.

10197 POLLERR An error has occurred (*revents* only).

10198 POLLHUP Device has been disconnected (*revents* only).

10199 POLLNVAL Invalid *fd* member (*revents* only).

10200 The significance and semantics of normal, priority, and high-priority data are file and device-
10201 specific.

10202 The following shall be declared as a function and may also be defined as a macro. A function
10203 prototype shall be provided.

10204 int poll(struct pollfd[], nfds_t, int);

10205 **APPLICATION USAGE**

10206 None.

10207 **RATIONALE**

10208 None.

10209 **FUTURE DIRECTIONS**

10210 None.

10211 **SEE ALSO**

10212 The System Interfaces volume of IEEE Std 1003.1-2001, *confstr()*, *poll()*, the Shell and Utilities
10213 volume of IEEE Std 1003.1-2001, *getconf*

10214 **CHANGE HISTORY**

10215 First released in Issue 4, Version 2.

10216 **Issue 6**10217 The description of the symbolic constants is updated to match the *poll()* function.10218 Text related to STREAMS has been moved to the *poll()* reference page.

10219 A note is added to the DESCRIPTION regarding the significance and semantics of normal,
10220 priority, and high-priority data.

10221 **NAME**

10222 pthread.h — threads

10223 **SYNOPSIS**

10224 THR #include <pthread.h>

10225

10226 **DESCRIPTION**

10227 The <pthread.h> header shall define the following symbols:

10228 BAR PTHREAD_BARRIER_SERIAL_THREAD

10229 PTHREAD_CANCEL_ASYNCHRONOUS

10230 PTHREAD_CANCEL_ENABLE

10231 PTHREAD_CANCEL_DEFERRED

10232 PTHREAD_CANCEL_DISABLE

10233 PTHREAD_CANCELED

10234 PTHREAD_COND_INITIALIZER

10235 PTHREAD_CREATE_DETACHED

10236 PTHREAD_CREATE_JOINABLE

10237 PTHREAD_EXPLICIT_SCHED

10238 PTHREAD_INHERIT_SCHED

10239 XSI PTHREAD_MUTEX_DEFAULT

10240 PTHREAD_MUTEX_ERRORCHECK

10241 PTHREAD_MUTEX_INITIALIZER

10242 XSI PTHREAD_MUTEX_NORMAL

10243 PTHREAD_MUTEX_RECURSIVE

10244 PTHREAD_ONCE_INIT

10245 TPI PTHREAD_PRIO_INHERIT

10246 TPP | TPI PTHREAD_PRIO_NONE

10247 TPP PTHREAD_PRIO_PROTECT

10248 PTHREAD_PROCESS_SHARED

10249 PTHREAD_PROCESS_PRIVATE

10250 TPS PTHREAD_SCOPE_PROCESS

10251 PTHREAD_SCOPE_SYSTEM

10252

10253 The following types shall be defined as described in <sys/types.h>:

10254 pthread_attr_t

10255 BAR pthread_barrier_t

10256 pthread_barrierattr_t

10257 pthread_cond_t

10258 pthread_condattr_t

10259 pthread_key_t

10260 pthread_mutex_t

10261 pthread_mutexattr_t

10262 pthread_once_t

10263 pthread_rwlock_t

10264 pthread_rwlockattr_t

10265 SPI pthread_spinlock_t

10266 pthread_t

10267 The following shall be declared as functions and may also be defined as macros. Function
 10268 prototypes shall be provided.

```

10269     int    pthread_atfork(void (*)(void), void (*)(void),
10270                          void (*)(void));
10271     int    pthread_attr_destroy(pthread_attr_t *);
10272     int    pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10273 XSI     int    pthread_attr_getguardsize(const pthread_attr_t *restrict,
10274                                         size_t *restrict);
10275 TPS     int    pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10276                                             int *restrict);
10277     int    pthread_attr_getschedparam(const pthread_attr_t *restrict,
10278                                       struct sched_param *restrict);
10279 TPS     int    pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10280                                            int *restrict);
10281 TPS     int    pthread_attr_getscope(const pthread_attr_t *restrict,
10282                                      int *restrict);
10283 TSA TSS int    pthread_attr_getstack(const pthread_attr_t *restrict,
10284                                       void **restrict, size_t *restrict);
10285 TSA     int    pthread_attr_getstackaddr(const pthread_attr_t *restrict,
10286                                           void **restrict);
10287 TSS     int    pthread_attr_getstacksize(const pthread_attr_t *restrict,
10288                                           size_t *restrict);
10289     int    pthread_attr_init(pthread_attr_t *);
10290     int    pthread_attr_setdetachstate(pthread_attr_t *, int);
10291 XSI     int    pthread_attr_setguardsize(pthread_attr_t *, size_t);
10292 TPS     int    pthread_attr_setinheritsched(pthread_attr_t *, int);
10293     int    pthread_attr_setschedparam(pthread_attr_t *restrict,
10294                                       const struct sched_param *restrict);
10295 TPS     int    pthread_attr_setschedpolicy(pthread_attr_t *, int);
10296     int    pthread_attr_setscope(pthread_attr_t *, int);
10297 TSA TSS int    pthread_attr_setstack(pthread_attr_t *, void *, size_t);
10298 TSA     int    pthread_attr_setstackaddr(pthread_attr_t *, void *);
10299 TSS     int    pthread_attr_setstacksize(pthread_attr_t *, size_t);
10300 BAR     int    pthread_barrier_destroy(pthread_barrier_t *);
10301     int    pthread_barrier_init(pthread_barrier_t *restrict,
10302                                const pthread_barrierattr_t *restrict, unsigned);
10303     int    pthread_barrier_wait(pthread_barrier_t *);
10304     int    pthread_barrierattr_destroy(pthread_barrierattr_t *);
10305 BAR TSH int    pthread_barrierattr_getpshared(
10306          const pthread_barrierattr_t *restrict, int *restrict);
10307 BAR     int    pthread_barrierattr_init(pthread_barrierattr_t *);
10308 BAR TSH int    pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10309     int    pthread_cancel(pthread_t);
10310     void    pthread_cleanup_push(void (*)(void *), void *);
10311     void    pthread_cleanup_pop(int);
10312     int    pthread_cond_broadcast(pthread_cond_t *);
10313     int    pthread_cond_destroy(pthread_cond_t *);
10314     int    pthread_cond_init(pthread_cond_t *restrict,
10315                             const pthread_condattr_t *restrict);
10316     int    pthread_cond_signal(pthread_cond_t *);
10317     int    pthread_cond_timedwait(pthread_cond_t *restrict,
10318                                   pthread_mutex_t *restrict, const struct timespec *restrict);
10319     int    pthread_cond_wait(pthread_cond_t *restrict,
10320                             pthread_mutex_t *restrict);

```



```

10321 int pthread_condattr_destroy(pthread_condattr_t *);
10322 CS int pthread_condattr_getclock(const pthread_condattr_t *restrict,
10323 clockid_t *restrict);
10324 TSH int pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10325 int *restrict);
10326 int pthread_condattr_init(pthread_condattr_t *);
10327 CS int pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
10328 TSH int pthread_condattr_setpshared(pthread_condattr_t *, int);
10329 int pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10330 void *(*)(void *), void *restrict);
10331 int pthread_detach(pthread_t);
10332 int pthread_equal(pthread_t, pthread_t);
10333 void pthread_exit(void *);
10334 XSI int pthread_getconcurrency(void);
10335 TCT int pthread_getcpuclockid(pthread_t, clockid_t *);
10336 TPS int pthread_getschedparam(pthread_t, int *restrict,
10337 struct sched_param *restrict);
10338 void *pthread_getspecific(pthread_key_t);
10339 int pthread_join(pthread_t, void **);
10340 int pthread_key_create(pthread_key_t *, void (*)(void *));
10341 int pthread_key_delete(pthread_key_t);
10342 int pthread_mutex_destroy(pthread_mutex_t *);
10343 TPP int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10344 int *restrict);
10345 int pthread_mutex_init(pthread_mutex_t *restrict,
10346 const pthread_mutexattr_t *restrict);
10347 int pthread_mutex_lock(pthread_mutex_t *);
10348 TPP int pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10349 int *restrict);
10350 TMO int pthread_mutex_timedlock(pthread_mutex_t *,
10351 const struct timespec *);
10352 int pthread_mutex_trylock(pthread_mutex_t *);
10353 int pthread_mutex_unlock(pthread_mutex_t *);
10354 int pthread_mutexattr_destroy(pthread_mutexattr_t *);
10355 TPP int pthread_mutexattr_getprioceiling(
10356 const pthread_mutexattr_t *restrict, int *restrict);
10357 TPP|TPI int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10358 int *restrict);
10359 TSH int pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10360 int *restrict);
10361 XSI int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10362 int *restrict);
10363 int pthread_mutexattr_init(pthread_mutexattr_t *);
10364 TPP int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10365 TPP|TPI int pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10366 TSH int pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10367 XSI int pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10368 int pthread_once(pthread_once_t *, void (*)(void));
10369 int pthread_rwlock_destroy(pthread_rwlock_t *);
10370 int pthread_rwlock_init(pthread_rwlock_t *restrict,
10371 const pthread_rwlockattr_t *restrict);
10372 int pthread_rwlock_rdlock(pthread_rwlock_t *);

```

```

10373 TMO      int      pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10374              const struct timespec *restrict);
10375      int      pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
10376              const struct timespec *restrict);
10377      int      pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10378      int      pthread_rwlock_trywrlock(pthread_rwlock_t *);
10379      int      pthread_rwlock_unlock(pthread_rwlock_t *);
10380      int      pthread_rwlock_wrlock(pthread_rwlock_t *);
10381      int      pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10382 TSH      int      pthread_rwlockattr_getpshared(
10383              const pthread_rwlockattr_t *restrict, int *restrict);
10384      int      pthread_rwlockattr_init(pthread_rwlockattr_t *);
10385 TSH      int      pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10386      pthread_t
10387              pthread_self(void);
10388      int      pthread_setcancelstate(int, int *);
10389      int      pthread_setcanceltype(int, int *);
10390 XSI      int      pthread_setconcurrency(int);
10391 TPS      int      pthread_setschedparam(pthread_t, int,
10392              const struct sched_param *);
10393 TPS      int      pthread_setschedprio(pthread_t, int);
10394      int      pthread_setspecific(pthread_key_t, const void *);
10395 SPI      int      pthread_spin_destroy(pthread_spinlock_t *);
10396      int      pthread_spin_init(pthread_spinlock_t *, int);
10397      int      pthread_spin_lock(pthread_spinlock_t *);
10398      int      pthread_spin_trylock(pthread_spinlock_t *);
10399      int      pthread_spin_unlock(pthread_spinlock_t *);
10400      void     pthread_testcancel(void);

```

10401 Inclusion of the <pthread.h> header shall make symbols defined in the headers <sched.h> and
10402 <time.h> visible.

10403 APPLICATION USAGE

10404 None.

10405 RATIONALE

10406 None.

10407 FUTURE DIRECTIONS

10408 None.

10409 SEE ALSO

10410 <sched.h>, <sys/types.h>, <time.h>, the System Interfaces volume of IEEE Std 1003.1-2001,
10411 pthread_attr_getguardsize(), pthread_attr_init(), pthread_attr_setscope(), pthread_barrier_destroy(),
10412 pthread_barrier_init(), pthread_barrier_wait(), pthread_barrierattr_destroy(),
10413 pthread_barrierattr_getpshared(), pthread_barrierattr_init(), pthread_barrierattr_setpshared(),
10414 pthread_cancel(), pthread_cleanup_pop(), pthread_cond_init(), pthread_cond_signal(),
10415 pthread_cond_wait(), pthread_condattr_getclock(), pthread_condattr_init(),
10416 pthread_condattr_setclock(), pthread_create(), pthread_detach(), pthread_equal(), pthread_exit(),
10417 pthread_getconcurrency(), pthread_getcpuclockid(), pthread_getschedparam(), pthread_join(),
10418 pthread_key_create(), pthread_key_delete(), pthread_mutex_init(), pthread_mutex_lock(),
10419 pthread_mutex_setprioceiling(), pthread_mutex_timedlock(), pthread_mutexattr_init(),
10420 pthread_mutexattr_gettype(), pthread_mutexattr_setprotocol(), pthread_once(),
10421 pthread_rwlock_destroy(), pthread_rwlock_init(), pthread_rwlock_rdlock(),
10422 pthread_rwlock_timedrdlock(), pthread_rwlock_timedwrlock(), pthread_rwlock_tryrdlock(),

10423 *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*, *pthread_rwlock_wrlock()*,
 10424 *pthread_rwlockattr_destroy()*, *pthread_rwlockattr_getpshared()*, *pthread_rwlockattr_init()*,
 10425 *pthread_rwlockattr_setpshared()*, *pthread_self()*, *pthread_setcancelstate()*, *pthread_setspecific()*,
 10426 *pthread_spin_destroy()*, *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*,
 10427 *pthread_spin_unlock()*

10428 CHANGE HISTORY

10429 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

10430 Issue 6

10431 The RTT margin markers are broken out into their POSIX options.

10432 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the
 10433 *pthread_cond_wait()* function.

10434 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the
 10435 *pthread_setschedparam()* function so that its second argument is of type **int**.

10436 The *pthread_getcpuclockid()* and *pthread_mutex_timedlock()* functions are added for alignment
 10437 with IEEE Std 1003.1d-1999.

10438 The following functions are added for alignment with IEEE Std 1003.1j-2000:

10439 *pthread_barrier_destroy()*, *pthread_barrier_init()*, *pthread_barrier_wait()*,
 10440 *pthread_barrierattr_destroy()*, *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*,
 10441 *pthread_barrierattr_setpshared()*, *pthread_condattr_getclock()*, *pthread_condattr_setclock()*,
 10442 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_spin_destroy()*,
 10443 *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*, and *pthread_spin_unlock()*.

10444 PTHREAD_RWLOCK_INITIALIZER is deleted for alignment with IEEE Std 1003.1j-2000.

10445 Functions previously marked as part of the Read-Write Locks option are now moved to the
 10446 Threads option.

10447 The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize()*,
 10448 *pthread_attr_getinheritsched()*, *pthread_attr_getschedparam()*, *pthread_attr_getschedpolicy()*,
 10449 *pthread_attr_getscope()*, *pthread_attr_getstackaddr()*, *pthread_attr_getstacksize()*,
 10450 *pthread_attr_setschedparam()*, *pthread_barrier_init()*, *pthread_barrierattr_getpshared()*,
 10451 *pthread_cond_init()*, *pthread_cond_signal()*, *pthread_cond_timedwait()*, *pthread_cond_wait()*,
 10452 *pthread_condattr_getclock()*, *pthread_condattr_getpshared()*, *pthread_create()*,
 10453 *pthread_getschedparam()*, *pthread_mutex_getprioceiling()*, *pthread_mutex_init()*,
 10454 *pthread_mutex_setprioceiling()*, *pthread_mutexattr_getprioceiling()*, *pthread_mutexattr_getprotocol()*,
 10455 *pthread_mutexattr_getpshared()*, *pthread_mutexattr_gettype()*, *pthread_rwlock_init()*,
 10456 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlockattr_getpshared()*, and
 10457 *pthread_sigmask()*.

10458 IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and
 10459 <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI
 10460 extension.

10461 IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for
 10462 the *pthread_kill()* and *pthread_sigmask()* functions. These are required to be in the <signal.h>
 10463 header. They are allowed here through the name space rules.

10464 IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread_setschedprio()* function.

10465 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/13 is applied, correcting shading errors
 10466 that were in contradiction with the System Interfaces volume of IEEE Std 1003.1-2001.

10467 **NAME**

10468 pwd.h — password structure

10469 **SYNOPSIS**

10470 #include <pwd.h>

10471 **DESCRIPTION**10472 The <pwd.h> header shall provide a definition for **struct passwd**, which shall include at least the
10473 following members:

10474	char	*pw_name	User's login name.
10475	uid_t	pw_uid	Numerical user ID.
10476	gid_t	pw_gid	Numerical group ID.
10477	char	*pw_dir	Initial working directory.
10478	char	*pw_shell	Program to use as shell.

10479 The **gid_t** and **uid_t** types shall be defined as described in <sys/types.h>.10480 The following shall be declared as functions and may also be defined as macros. Function
10481 prototypes shall be provided.

```

10482     struct passwd *getpwnam(const char *);
10483     struct passwd *getpwuid(uid_t);
10484 TSF   int         getpwnam_r(const char *, struct passwd *, char *,
10485                               size_t, struct passwd **);
10486     int         getpwuid_r(uid_t, struct passwd *, char *,
10487                               size_t, struct passwd **);
10488 XSI   void         endpwent(void);
10489     struct passwd *getpwent(void);
10490     void         setpwent(void);
10491

```

10492 **APPLICATION USAGE**

10493 None.

10494 **RATIONALE**

10495 None.

10496 **FUTURE DIRECTIONS**

10497 None.

10498 **SEE ALSO**10499 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *endpwent()*, *getpwnam()*,
10500 *getpwuid()*10501 **CHANGE HISTORY**

10502 First released in Issue 1.

10503 **Issue 5**

10504 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

10505 **Issue 6**10506 The following new requirements on POSIX implementations derive from alignment with the
10507 Single UNIX Specification:

- 10508 • The **gid_t** and **uid_t** types are mandated.
- 10509 • The *getpwnam_r()* and *getpwuid_r()* functions are marked as part of the Thread-Safe
10510 Functions option.

10511 **NAME**

10512 regex.h — regular expression matching types

10513 **SYNOPSIS**

10514 #include <regex.h>

10515 **DESCRIPTION**

10516 The <regex.h> header shall define the structures and symbolic constants used by the *regcomp()*,
10517 *regex()*, *regerror()*, and *regfree()* functions.

10518 The structure type **regex_t** shall contain at least the following member:

10519 size_t re_nsub Number of parenthesized subexpressions.

10520 The type **size_t** shall be defined as described in <sys/types.h>.

10521 The type **regoff_t** shall be defined as a signed integer type that can hold the largest value that
10522 can be stored in either a type **off_t** or type **ssize_t**. The structure type **regmatch_t** shall contain
10523 at least the following members:

10524	regoff_t	rm_so	Byte offset from start of string
10525			to start of substring.
10526	regoff_t	rm_eo	Byte offset from start of string of the
10527			first character after the end of substring.

10528 Values for the *cflags* parameter to the *regcomp()* function are as follows:

10529 REG_EXTENDED Use Extended Regular Expressions.

10530 REG_ICASE Ignore case in match.

10531 REG_NOSUB Report only success or fail in *regex()*.

10532 REG_NEWLINE Change the handling of <newline>.

10533 Values for the *eflags* parameter to the *regex()* function are as follows:

10534 REG_NOTBOL The circumflex character ('^'), when taken as a special character, does
10535 not match the beginning of *string*.

10536 REG_NOTEOL The dollar sign ('\$'), when taken as a special character, does not match
10537 the end of *string*.

10538 The following constants shall be defined as error return values:

10539 REG_NOMATCH *regex()* failed to match.

10540 REG_BADPAT Invalid regular expression.

10541 REG_ECOLLATE Invalid collating element referenced.

10542 REG_ETYPE Invalid character class type referenced.

10543 REG_EESCAPE Trailing '\\' in pattern.

10544 REG_ESUBREG Number in *\digit* invalid or in error.

10545 REG_EBRACK "[]" imbalance.

10546 REG_EPAREN "\" "(" imbalance.

10547 REG_EBRACE "\" "{" imbalance.

10548 REG_BADBR Content of "\" "{" invalid: not a number, number too large, more than
10549 two numbers, first larger than second.

10550 REG_ERANGE Invalid endpoint in range expression.

10551 REG_ESPACE Out of memory.

10552 REG_BADRPT '?', '*', or '+' not preceded by valid regular expression.

10553 OB REG_ENOSYS Reserved.

10554 The following shall be declared as functions and may also be defined as macros. Function
10555 prototypes shall be provided.

```
10556 int      regcomp(regex_t *restrict, const char *restrict, int);
10557 size_t  regerror(int, const regex_t *restrict, char *restrict, size_t);
10558 int      regexec(const regex_t *restrict, const char *restrict, size_t,
10559                 regmatch_t[restrict], int);
10560 void     regfree(regex_t *);
```

10561 The implementation may define additional macros or constants using names beginning with
10562 REG_.

10563 **APPLICATION USAGE**

10564 None.

10565 **RATIONALE**

10566 None.

10567 **FUTURE DIRECTIONS**

10568 None.

10569 **SEE ALSO**

10570 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *regcomp()*, the Shell and
10571 Utilities volume of IEEE Std 1003.1-2001

10572 **CHANGE HISTORY**

10573 First released in Issue 4.

10574 Originally derived from the ISO POSIX-2 standard.

10575 **Issue 6**

10576 The REG_ENOSYS constant is marked obsolescent.

10577 The **restrict** keyword is added to the prototypes for *regcomp()*, *regerror()*, and *regexec()*.

10578 A statement is added that the **size_t** type is defined as described in <sys/types.h>.

10579 **NAME**10580 sched.h — execution scheduling (**REALTIME**)10581 **SYNOPSIS**

10582 PS #include <sched.h>

10583

10584 **DESCRIPTION**

10585 The <sched.h> header shall define the **sched_param** structure, which contains the scheduling
 10586 parameters required for implementation of each supported scheduling policy. This structure
 10587 shall contain at least the following member:

10588 int sched_priority Process execution scheduling priority.

10589 SS|TSP In addition, if **_POSIX_SPORADIC_SERVER** or **_POSIX_THREAD_SPORADIC_SERVER** is
 10590 defined, the **sched_param** structure defined in <sched.h> shall contain the following members
 10591 in addition to those specified above:

10592	int	sched_ss_low_priority	Low scheduling priority for
10593			sporadic server.
10594	struct timespec	sched_ss_repl_period	Replenishment period for
10595			sporadic server.
10596	struct timespec	sched_ss_init_budget	Initial budget for sporadic server.
10597	int	sched_ss_max_repl	Maximum pending replenishments for
10598			sporadic server.

10599

10600 Each process is controlled by an associated scheduling policy and priority. Associated with each
 10601 policy is a priority range. Each policy definition specifies the minimum priority range for that
 10602 policy. The priority ranges for each policy may overlap the priority ranges of other policies.

10603 Four scheduling policies are defined; others may be defined by the implementation. The four
 10604 standard policies are indicated by the values of the following symbolic constants:

10605 **SCHED_FIFO** First in-first out (FIFO) scheduling policy.

10606 **SCHED_RR** Round robin scheduling policy.

10607 SS|TSP **SCHED_SPORADIC** Sporadic server scheduling policy.

10608 **SCHED_OTHER** Another scheduling policy.

10609 The values of these constants are distinct.

10610 The following shall be declared as functions and may also be defined as macros. Function
 10611 prototypes shall be provided.

10612	TPS	int	sched_get_priority_max(int);	
10613		int	sched_get_priority_min(int);	
10614		int	sched_getparam(pid_t, struct sched_param *);	
10615		int	sched_getscheduler(pid_t);	
10616	TPS	int	sched_rr_get_interval(pid_t, struct timespec *);	
10617		int	sched_setparam(pid_t, const struct sched_param *);	
10618		int	sched_setscheduler(pid_t, int, const struct sched_param *);	
10619	THR	int	sched_yield(void);	
10620				

10621 Inclusion of the <sched.h> header may make visible all symbols from the <time.h> header.

10622 **APPLICATION USAGE**

10623 None.

10624 **RATIONALE**

10625 None.

10626 **FUTURE DIRECTIONS**

10627 None.

10628 **SEE ALSO**

10629 <time.h>

10630 **CHANGE HISTORY**

10631 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10632 **Issue 6**

10633 The <sched.h> header is marked as part of the Process Scheduling option.

10634 Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC
 10635 scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

10636 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose
 10637 members *sched_ss_repl_period* and *sched_ss_init_budget* should be type **struct timespec** and not
 10638 **timespec**.

10639 Symbols from <time.h> may be made visible when <sched.h> is included.

10640 IEEE Std 1003.1-2001/Cor 1-2002, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied, |
 10641 aligning the function prototype shading and margin codes with the System Interfaces volume of |
 10642 IEEE Std 1003.1-2001. |

10643 **NAME**

10644 search.h — search tables

10645 **SYNOPSIS**

10646 XSI `#include <search.h>`

10647

10648 **DESCRIPTION**

10649 The <search.h> header shall define the **ENTRY** type for structure **entry** which shall include the
10650 following members:

10651 char *key
10652 void *data

10653 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as
10654 follows:

10655 enum { FIND, ENTER } ACTION;
10656 enum { preorder, postorder, endorder, leaf } VISIT;

10657 The **size_t** type shall be defined as described in <sys/types.h>.

10658 The following shall be declared as functions and may also be defined as macros. Function
10659 prototypes shall be provided.

```
10660 int      hcreate(size_t);
10661 void     hdestroy(void);
10662 ENTRY *hsearch(ENTRY, ACTION);
10663 void     insque(void *, void *);
10664 void     *lfind(const void *, const void *, size_t *,
10665               size_t, int (*)(const void *, const void *));
10666 void     *lsearch(const void *, void *, size_t *,
10667               size_t, int (*)(const void *, const void *));
10668 void     remque(void *);
10669 void     *tdelete(const void *restrict, void **restrict,
10670               int (*)(const void *, const void *));
10671 void     *tfind(const void *, void *const *,
10672               int (*)(const void *, const void *));
10673 void     *tsearch(const void *, void **,
10674               int (*)(const void *, const void *));
10675 void     twalk(const void *,
10676               void (*)(const void *, VISIT, int ));
```

10677 **APPLICATION USAGE**

10678 None.

10679 **RATIONALE**

10680 None.

10681 **FUTURE DIRECTIONS**

10682 None.

10683 **SEE ALSO**

10684 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *hcreate()*, *insque()*,
10685 *lsearch()*, *remque()*, *tsearch()*

10686 **CHANGE HISTORY**

10687 First released in Issue 1. Derived from Issue 1 of the SVID.

10688 **Issue 6**

10689 The Open Group Corrigendum U021/6 is applied, updating the prototypes for *tdelete()* and
10690 *tsearch()*.

10691 The **restrict** keyword is added to the prototype for *tdelete()*.

10692 **NAME**10693 semaphore.h — semaphores (**REALTIME**)10694 **SYNOPSIS**10695 SEM `#include <semaphore.h>`

10696

10697 **DESCRIPTION**

10698 The <semaphore.h> header shall define the **sem_t** type, used in performing semaphore
 10699 operations. The semaphore may be implemented using a file descriptor, in which case
 10700 applications are able to open up at least a total of {OPEN_MAX} files and semaphores. The
 10701 symbol SEM_FAILED shall be defined (see *sem_open()*).

10702 The following shall be declared as functions and may also be defined as macros. Function
 10703 prototypes shall be provided.

```
10704     int      sem_close(sem_t *);
10705     int      sem_destroy(sem_t *);
10706     int      sem_getvalue(sem_t *restrict, int *restrict);
10707     int      sem_init(sem_t *, int, unsigned);
10708     sem_t *sem_open(const char *, int, ...);
10709     int      sem_post(sem_t *);
10710 TMO     int      sem_timedwait(sem_t *restrict, const struct timespec *restrict);
10711     int      sem_trywait(sem_t *);
10712     int      sem_unlink(const char *);
10713     int      sem_wait(sem_t *);
```

10714 Inclusion of the <semaphore.h> header may make visible symbols defined in the headers
 10715 <fcntl.h> and <sys/types.h>.

10716 **APPLICATION USAGE**

10717 None.

10718 **RATIONALE**

10719 None.

10720 **FUTURE DIRECTIONS**

10721 None.

10722 **SEE ALSO**

10723 <fcntl.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *sem_destroy()*,
 10724 *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*,
 10725 *sem_wait()*

10726 **CHANGE HISTORY**

10727 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10728 **Issue 6**

10729 The <semaphore.h> header is marked as part of the Semaphores option.

10730 The Open Group Corrigendum U021/3 is applied, adding a description of SEM_FAILED.

10731 The *sem_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.10732 The **restrict** keyword is added to the prototypes for *sem_getvalue()* and *sem_timedwait()*.

10733 **NAME**

10734 setjmp.h — stack environment declarations

10735 **SYNOPSIS**

10736 #include <setjmp.h>

10737 **DESCRIPTION**

10738 CX Some of the functionality described on this reference page extends the ISO C standard.
 10739 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 10740 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 10741 symbols in this header.

10742 CX The <setjmp.h> header shall define the array types **jmp_buf** and **sigjmp_buf**.

10743 The following shall be declared as functions and may also be defined as macros. Function
 10744 prototypes shall be provided.

```
10745       void    longjmp(jmp_buf, int);
10746 CX       void   siglongjmp(sigjmp_buf, int);
10747 XSI       void   _longjmp(jmp_buf, int);
```

10748

10749 The following may be declared as a function, or defined as a macro, or both. Function prototypes
 10750 shall be provided.

```
10751       int     setjmp(jmp_buf);
10752 CX       int     sigsetjmp(sigjmp_buf, int);
10753 XSI       int     _setjmp(jmp_buf);
```

10754

10755 **APPLICATION USAGE**

10756 None.

10757 **RATIONALE**

10758 None.

10759 **FUTURE DIRECTIONS**

10760 None.

10761 **SEE ALSO**

10762 The System Interfaces volume of IEEE Std 1003.1-2001, *longjmp()*, *_longjmp()*, *setjmp()*,
 10763 *siglongjmp()*, *sigsetjmp()*

10764 **CHANGE HISTORY**

10765 First released in Issue 1.

10766 **Issue 6**

10767 Extensions beyond the ISO C standard are marked.

10768 **NAME**

10769 signal.h — signals

10770 **SYNOPSIS**

10771 #include <signal.h>

10772 **DESCRIPTION**

10773 CX Some of the functionality described on this reference page extends the ISO C standard.
10774 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
10775 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
10776 symbols in this header.

10777 The <signal.h> header shall define the following symbolic constants, each of which expands to a
10778 distinct constant expression of the type:

10779 void (*)(int)

10780 whose value matches no declarable function.

10781 SIG_DFL Request for default signal handling.

10782 SIG_ERR Return value from *signal()* in case of error.

10783 CX SIG_HOLD Request that signal be held.

10784 SIG_IGN Request that signal be ignored.

10785 The following data types shall be defined through **typedef**:

10786 **sig_atomic_t** Possibly volatile-qualified integer type of an object that can be accessed as
10787 an atomic entity, even in the presence of asynchronous interrupts.

10788 CX **sigset_t** Integer or structure type of an object used to represent sets of signals.

10789 CX **pid_t** As described in <sys/types.h>.

10790 RTS The <signal.h> header shall define the **sigevent** structure, which has at least the following
10791 members:

10792	int	sigev_notify	Notification type.
10793	int	sigev_signo	Signal number.
10794	union sigval	sigev_value	Signal value.
10795	void(*) (union sigval)	sigev_notify_function	Notification function.
10796	(pthread_attr_t *)	sigev_notify_attributes	Notification attributes.

10797 The following values of *sigev_notify* shall be defined:

10798 SIGEV_NONE No asynchronous notification is delivered when the event of interest
10799 occurs.

10800 SIGEV_SIGNAL A queued signal, with an application-defined value, is generated when
10801 the event of interest occurs.

10802 SIGEV_THREAD A notification function is called to perform notification.

10803 The **sigval** union shall be defined as:

10804	int	sival_int	Integer signal value.
10805	void	*sival_ptr	Pointer signal value.

10806 This header shall also declare the macros SIGRTMIN and SIGRTMAX, which evaluate to integer
10807 expressions, and specify a range of signal numbers that are reserved for application use and for
10808 which the realtime signal behavior specified in this volume of IEEE Std 1003.1-2001 is supported.

10809 The signal numbers in this range do not overlap any of the signals specified in the following
10810 table.

10811 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal
10812 numbers.

10813 It is implementation-defined whether realtime signal behavior is supported for other signals.

10814 This header also declares the constants that are used to refer to the signals that occur in the
10815 system. Signals defined here begin with the letters SIG. Each of the signals have distinct positive
10816 integer values. The value 0 is reserved for use as the null signal (see *kill()*). Additional
10817 implementation-defined signals may occur in the system.

10818 CX The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT,
10819 SIGSEGV, and SIGTERM to be defined.

10820 The following signals shall be supported on all implementations (default actions are explained
10821 below the table):

Signal	Default Action	Description
SIGABRT	A	Process abort signal.
SIGALRM	T	Alarm clock.
SIGBUS	A	Access to an undefined portion of a memory object.
SIGCHLD	I	Child process terminated, stopped, or continued.
SIGCONT	C	Continue executing, if stopped.
SIGFPE	A	Erroneous arithmetic operation.
SIGHUP	T	Hangup.
SIGILL	A	Illegal instruction.
SIGINT	T	Terminal interrupt signal.
SIGKILL	T	Kill (cannot be caught or ignored).
SIGPIPE	T	Write on a pipe with no one to read it.
SIGQUIT	A	Terminal quit signal.
SIGSEGV	A	Invalid memory reference.
SIGSTOP	S	Stop executing (cannot be caught or ignored).
SIGTERM	T	Termination signal.
SIGTSTP	S	Terminal stop signal.
SIGTTIN	S	Background process attempting read.
SIGTTOU	S	Background process attempting write.
SIGUSR1	T	User-defined signal 1.
SIGUSR2	T	User-defined signal 2.
SIGPOLL	T	Pollable event.
SIGPROF	T	Profiling timer expired.
SIGSYS	A	Bad system call.
SIGTRAP	A	Trace/breakpoint trap.
SIGURG	I	High bandwidth data is available at a socket.
SIGVTALRM	T	Virtual timer expired.
SIGXCPU	A	CPU time limit exceeded.
SIGXFSZ	A	File size limit exceeded.

10853 The default actions are as follows:

10854 T Abnormal termination of the process. The process is terminated with all the consequences
10855 of *_exit()* except that the status made available to *wait()* and *waitpid()* indicates abnormal
10856 termination by the specified signal.

10857 A Abnormal termination of the process.
 10858 XSI Additionally, implementation-defined abnormal termination actions, such as creation of a
 10859 **core** file, may occur.
 10860 I Ignore the signal.
 10861 S Stop the process.
 10862 C Continue the process, if it is stopped; otherwise, ignore the signal.

10863 CX The header shall provide a declaration of **struct sigaction**, including at least the following
 10864 members:

10865	void (*sa_handler)(int)	Pointer to a signal-catching function or one of the macros
10866		SIG_IGN or SIG_DFL.
10867	sigset_t sa_mask	Set of signals to be blocked during execution of the signal
10868		handling function.
10869	int sa_flags	Special flags.
10870	void (*sa_sigaction)(int, siginfo_t *, void *)	
10871		Pointer to a signal-catching function.

10872

10873 XSI The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application
 10874 shall not use both simultaneously.

10875 The following shall be declared as constants:

10876 CX	SA_NOCLDSTOP	Do not generate SIGCHLD when children stop
10877 XSI		or stopped children continue.
10878 CX	SIG_BLOCK	The resulting set is the union of the current set and the signal set pointed
10879		to by the argument <i>set</i> .
10880 CX	SIG_UNBLOCK	The resulting set is the intersection of the current set and the complement
10881		of the signal set pointed to by the argument <i>set</i> .
10882 CX	SIG_SETMASK	The resulting set is the signal set pointed to by the argument <i>set</i> .
10883 XSI	SA_ONSTACK	Causes signal delivery to occur on an alternate stack.
10884 XSI	SA_RESETHAND	Causes signal dispositions to be set to SIG_DFL on entry to signal
10885		handlers.
10886 XSI	SA_RESTART	Causes certain functions to become restartable.
10887 XSI	SA_SIGINFO	Causes extra information to be passed to signal handlers at the time of
10888		receipt of a signal.
10889 XSI	SA_NOCLDWAIT	Causes implementations not to create zombie processes on child death.
10890 XSI	SA_NODEFER	Causes signal not to be automatically blocked on entry to signal handler.
10891 XSI	SS_ONSTACK	Process is executing on an alternate signal stack.
10892 XSI	SS_DISABLE	Alternate signal stack is disabled.
10893 XSI	MINSIGSTKSZ	Minimum stack size for a signal handler.
10894 XSI	SIGSTKSZ	Default size in bytes for the alternate signal stack.
10895 XSI	The ucontext_t structure shall be defined through typedef as described in <ucontext.h>.	
10896	The mcontext_t type shall be defined through typedef as described in <ucontext.h>.	

10897 The <signal.h> header shall define the **stack_t** type as a structure that includes at least the
 10898 following members:

10899	void	*ss_sp	Stack base or pointer.
10900	size_t	ss_size	Stack size.
10901	int	ss_flags	Flags.

10902 The <signal.h> header shall define the **sigstack** structure that includes at least the following
 10903 members:

10904	int	ss_onstack	Non-zero when signal stack is in use.
10905	void	*ss_sp	Signal stack pointer.

10906

10907 CX The <signal.h> header shall define the **siginfo_t** type as a structure that includes at least the
 10908 following members:

10909 CX	int	si_signo	Signal number.
10910 XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with this signal, as defined in <errno.h>.
10911			
10912 CX	int	si_code	Signal code.
10913 XSI	pid_t	si_pid	Sending process ID.
10914	uid_t	si_uid	Real user ID of sending process.
10915	void	*si_addr	Address of faulting instruction.
10916	int	si_status	Exit value or signal.
10917	long	si_band	Band event for SIGPOLL.
10918 RTS	union sigval	si_value	Signal value.
10919			

10920 The macros specified in the **Code** column of the following table are defined for use as values of
 10921 XSI *si_code* that are signal-specific or non-signal-specific reasons why the signal was generated.

10922

10923

10924 xSI

10925

10926

10927

10928

10929

10930

10931

10932

10933

10934

10935

10936

10937

10938

10939

10940

10941

10942

10943

10944

10945

10946

10947

10948

10949

10950

10951

10952

10953

10954

10955

10956

10957

10958

10959 CX

10960

10961

10962

10963

10964

10965

Signal	Code	Reason
SIGILL	ILL_ILLOPC	Illegal opcode.
	ILL_ILLOPN	Illegal operand.
	ILL_ILLADR	Illegal addressing mode.
	ILL_ILLTRP	Illegal trap.
	ILL_PRVOPC	Privileged opcode.
	ILL_PRVREG	Privileged register.
	ILL_COPROC	Coprocessor error.
	ILL_BADSTK	Internal stack error.
SIGFPE	FPE_INTDIV	Integer divide by zero.
	FPE_INTOVF	Integer overflow.
	FPE_FLTDIV	Floating-point divide by zero.
	FPE_FLTOVF	Floating-point overflow.
	FPE_FLTUND	Floating-point underflow.
	FPE_FLTRES	Floating-point inexact result.
	FPE_FLTINV	Invalid floating-point operation.
	FPE_FLTSUB	Subscript out of range.
SIGSEGV	SEGV_MAPERR	Address not mapped to object.
	SEGV_ACCERR	Invalid permissions for mapped object.
SIGBUS	BUS_ADRALN	Invalid address alignment.
	BUS_ADRERR	Nonexistent physical address.
	BUS_OBJERR	Object-specific hardware error.
SIGTRAP	TRAP_BRKPT	Process breakpoint.
	TRAP_TRACE	Process trace trap.
SIGCHLD	CLD_EXITED	Child has exited.
	CLD_KILLED	Child has terminated abnormally and did not create a core file.
	CLD_DUMPED	Child has terminated abnormally and created a core file.
	CLD_TRAPPED	Traced child has trapped.
	CLD_STOPPED	Child has stopped.
	CLD_CONTINUED	Stopped child has continued.
SIGPOLL	POLL_IN	Data input available.
	POLL_OUT	Output buffers available.
	POLL_MSG	Input message available.
	POLL_ERR	I/O error.
	POLL_PRI	High priority input available.
	POLL_HUP	Device disconnected.
Any	SI_USER	Signal sent by <i>kill()</i> .
	SI_QUEUE	Signal sent by the <i>sigqueue()</i> .
	SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .
	SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
	SI_MESGQ	Signal generated by arrival of a message on an empty message queue.

10966 xSI

10967

10968

10969

10970

Implementations may support additional *si_code* values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.

10971 In addition, the following signal-specific information shall be available:

Signal	Member	Value
SIGILL SIGFPE	void * <i>si_addr</i>	Address of faulting instruction.
SIGSEGV SIGBUS	void * <i>si_addr</i>	Address of faulting memory reference.
SIGCHLD	pid_t <i>si_pid</i> int <i>si_status</i> uid_t <i>si_uid</i>	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
SIGPOLL	long <i>si_band</i>	Band event for POLL_IN, POLL_OUT, or POLL_MSG.

10982 For some implementations, the value of *si_addr* may be inaccurate.

10983 The following shall be declared as functions and may also be defined as macros:

```

10984 XSI void (*bsd_signal(int, void (*)(int)))(int);
10985 CX   int kill(pid_t, int);
10986 XSI  int killpg(pid_t, int);
10987 THR  int pthread_kill(pthread_t, int);
10988     int pthread_sigmask(int, const sigset_t *, sigset_t *);
10989     int raise(int);
10990 CX   int sigaction(int, const struct sigaction *restrict,
10991                   struct sigaction *restrict);
10992     int sigaddset(sigset_t *, int);
10993 XSI  int sigaltstack(const stack_t *restrict, stack_t *restrict);
10994 CX   int sigdelset(sigset_t *, int);
10995     int sigemptyset(sigset_t *);
10996     int sigfillset(sigset_t *);
10997 XSI  int sighold(int);
10998     int sigignore(int);
10999     int siginterrupt(int, int);
11000 CX   int sigismember(const sigset_t *, int);
11001     void (*signal(int, void (*)(int)))(int);
11002 XSI  int sigpause(int);
11003 CX   int sigpending(sigset_t *);
11004     int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
11005 RTS  int sigqueue(pid_t, int, const union sigval);
11006 XSI  int sigrelse(int);
11007     void (*sigset(int, void (*)(int)))(int);
11008 CX   int sigsuspend(const sigset_t *);
11009 RTS  int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
11010                     const struct timespec *restrict);
11011 CX   int sigwait(const sigset_t *restrict, int *restrict);
11012 RTS  int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
11013

```

11014 CX Inclusion of the <signal.h> header may make visible all symbols from the <time.h> header.

11015 APPLICATION USAGE

11016 None.

11017 RATIONALE

11018 None.

11019 FUTURE DIRECTIONS

11020 None.

11021 SEE ALSO

11022 <errno.h>, <stropts.h>, <sys/types.h>, <time.h>, <ucontext.h>, the System Interfaces volume of
 11023 IEEE Std 1003.1-2001, *alarm()*, *bsd_signal()*, *ioctl()*, *kill()*, *killpg()*, *raise()*, *sigaction()*, *sigaddset()*,
 11024 *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *siginterrupt()*, *sigismember()*, *signal()*,
 11025 *sigpending()*, *sigprocmask()*, *sigqueue()*, *sigsuspend()*, *sigwaitinfo()*, *wait()*, *waitid()*

11026 CHANGE HISTORY

11027 First released in Issue 1.

11028 Issue 5

11029 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 11030 Threads Extension.

11031 The default action for SIGURG is changed from i to iii. The function prototype for *sigmask()* is
 11032 removed.

11033 Issue 6

11034 The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for
 11035 abnormal termination is clarified.

11036 The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()*
 11037 function.

11038 The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev_notify_function*
 11039 function member of the **sigevent** structure.

11040 The following new requirements on POSIX implementations derive from alignment with the
 11041 Single UNIX Specification:

11042 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now
 11043 mandated. This is also a FIPS requirement.

11044 • The **pid_t** definition is mandated.

11045 The RT markings are changed to RTS to denote that the semantics are part of the Realtime
 11046 Signals Extension option.

11047 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,
 11048 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

11049 IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from
 11050 <time.h> may be made visible when <signal.h> is included.

11051 Extensions beyond the ISO C standard are marked.

11052 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/14 is applied, changing the descriptive
 11053 text for members of **struct sigaction**.

11054 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/15 is applied, correcting the definition of
 11055 the *sa_sigaction* member of **struct sigaction**.

11056 **NAME**11057 spawn.h — spawn (**ADVANCED REALTIME**)11058 **SYNOPSIS**

11059 SPN #include <spawn.h>

11060

11061 **DESCRIPTION**

11062 The <spawn.h> header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t**
 11063 types used in performing spawn operations.

11064 The <spawn.h> header shall define the flags that may be set in a **posix_spawnattr_t** object using
 11065 the *posix_spawnattr_setflags()* function:

11066 POSIX_SPAWN_RESETPIDS

11067 POSIX_SPAWN_SETPGROUP

11068 PS POSIX_SPAWN_SETSCHEDPARAM

11069 POSIX_SPAWN_SETSCHEDULER

11070 POSIX_SPAWN_SETSIGDEF

11071 POSIX_SPAWN_SETSIGMASK

11072 The following shall be declared as functions and may also be defined as macros. Function
 11073 prototypes shall be provided.

```

11074 int    posix_spawn(pid_t *restrict, const char *restrict,
11075                  const posix_spawn_file_actions_t *,
11076                  const posix_spawnattr_t *restrict, char *const [restrict],
11077                  char *const [restrict]);
11078 int    posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11079                  int);
11080 int    posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11081                  int, int);
11082 int    posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11083                  int, const char *restrict, int, mode_t);
11084 int    posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11085 int    posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11086 int    posix_spawnattr_destroy(posix_spawnattr_t *);
11087 int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11088                  sigset_t *restrict);
11089 int    posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11090                  short *restrict);
11091 int    posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11092                  pid_t *restrict);
11093 PS int    posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11094                  struct sched_param *restrict);
11095 int    posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11096                  int *restrict);
11097 int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11098                  sigset_t *restrict);
11099 int    posix_spawnattr_init(posix_spawnattr_t *);
11100 int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11101                  const sigset_t *restrict);
11102 int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11103 int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);

```

```

11104 PS      int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
11105              const struct sched_param *restrict);
11106      int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
11107      int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
11108              const sigset_t *restrict);
11109      int    posix_spawn(pid_t *restrict, const char *restrict,
11110              const posix_spawn_file_actions_t *,
11111              const posix_spawnattr_t *restrict,
11112              char *const [restrict], char *const [restrict]);

```

11113 Inclusion of the <spawn.h> header may make visible symbols defined in the <sched.h>,
11114 <signal.h>, and <sys/types.h> headers.

11115 APPLICATION USAGE

11116 None.

11117 RATIONALE

11118 None.

11119 FUTURE DIRECTIONS

11120 None.

11121 SEE ALSO

11122 <sched.h>, <semaphore.h>, <signal.h>, <sys/types.h>, the System Interfaces volume of
11123 IEEE Std 1003.1-2001, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
11124 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
11125 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*, *posix_spawnattr_init()*,
11126 *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*, *posix_spawnattr_setpgroup()*,
11127 *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*,
11128 *posix_spawn()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*,
11129 *posix_spawn_file_actions_addopen()*, *posix_spawn_file_actions_destroy()*,
11130 *posix_spawn_file_actions_init()*, *posix_spawnnp()*

11131 CHANGE HISTORY

11132 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

11133 The **restrict** keyword is added to the prototypes for *posix_spawn()*,
11134 *posix_spawn_file_actions_addopen()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*,
11135 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
11136 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setschedparam()*,
11137 *posix_spawnattr_setsigmask()*, and *posix_spawnnp()*.

11138 **NAME**

11139 stdarg.h — handle variable argument list

11140 **SYNOPSIS**

11141 #include <stdarg.h>

```
11142       void va_start(va_list ap, argN);
11143       void va_copy(va_list dest, va_list src);
11144       type va_arg(va_list ap, type);
11145       void va_end(va_list ap);
```

11146 **DESCRIPTION**

11147 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11148 conflict between the requirements described here and the ISO C standard is unintentional. This
 11149 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11150 The <stdarg.h> header shall contain a set of macros which allows portable functions that accept
 11151 variable argument lists to be written. Functions that have variable argument lists (such as
 11152 *printf()*) but do not use these macros are inherently non-portable, as different systems use
 11153 different argument-passing conventions.

11154 The type **va_list** shall be defined for variables used to traverse the list.

11155 The *va_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to
 11156 *va_arg()*.

11157 The *va_copy()* macro initializes *dest* as a copy of *src*, as if the *va_start()* macro had been applied
 11158 to *dest* followed by the same sequence of uses of the *va_arg()* macro as had previously been used
 11159 to reach the present state of *src*. Neither the *va_copy()* nor *va_start()* macro shall be invoked to
 11160 reinitialize *dest* without an intervening invocation of the *va_end()* macro for the same *dest*.

11161 The object *ap* may be passed as an argument to another function; if that function invokes the
 11162 *va_arg()* macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall
 11163 be passed to the *va_end()* macro prior to any further reference to *ap*. The parameter *argN* is the
 11164 identifier of the rightmost parameter in the variable parameter list in the function definition (the
 11165 one just before the ...). If the parameter *argN* is declared with the **register** storage class, with a
 11166 function type or array type, or with a type that is not compatible with the type that results after
 11167 application of the default argument promotions, the behavior is undefined.

11168 The *va_arg()* macro shall return the next argument in the list pointed to by *ap*. Each invocation
 11169 of *va_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*
 11170 parameter shall be a type name specified such that the type of a pointer to an object that has the
 11171 specified type can be obtained simply by postfixing a '*' to type. If there is no actual next
 11172 argument, or if *type* is not compatible with the type of the actual next argument (as promoted
 11173 according to the default argument promotions), the behavior is undefined, except for the
 11174 following cases:

- 11175 • One type is a signed integer type, the other type is the corresponding unsigned integer type,
 11176 and the value is representable in both types.
- 11177 • One type is a pointer to **void** and the other is a pointer to a character type.
- 11178 XSI • Both types are pointers.

11179 Different types can be mixed, but it is up to the routine to know what type of argument is
 11180 expected.

11181 The *va_end()* macro is used to clean up; it invalidates *ap* for use (unless *va_start()* or *va_copy()* is
 11182 invoked again).

11183 Each invocation of the *va_start()* and *va_copy()* macros shall be matched by a corresponding
11184 invocation of the *va_end()* macro in the same function.

11185 Multiple traversals, each bracketed by *va_start()*... *va_end()*, are possible.

11186 **EXAMPLES**

11187 This example is a possible implementation of *execl()*:

```
11188 #include <stdarg.h>
11189 #define MAXARGS 31
11190 /*
11191  * execl is called by
11192  * execl(file, arg1, arg2, ..., (char *) (0));
11193  */
11194 int execl(const char *file, const char *args, ...)
11195 {
11196     va_list ap;
11197     char *array[MAXARGS + 1];
11198     int argno = 0;
11199
11200     va_start(ap, args);
11201     while (args != 0 && argno < MAXARGS)
11202     {
11203         array[argno++] = args;
11204         args = va_arg(ap, const char *);
11205     }
11206     array[argno] = (char *) 0;
11207     va_end(ap);
11208     return execv(file, array);
11209 }
```

11209 **APPLICATION USAGE**

11210 It is up to the calling routine to communicate to the called routine how many arguments there
11211 are, since it is not always possible for the called routine to determine this in any other way. For
11212 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell
11213 how many arguments are there by the *format* argument.

11214 **RATIONALE**

11215 None.

11216 **FUTURE DIRECTIONS**

11217 None.

11218 **SEE ALSO**

11219 The System Interfaces volume of IEEE Std 1003.1-2001, *exec*, *printf()*

11220 **CHANGE HISTORY**

11221 First released in Issue 4. Derived from the ANSI C standard.

11222 **Issue 6**

11223 This reference page is updated to align with the ISO / IEC 9899:1999 standard.

11224 NAME

11225 stdbool.h — boolean type and values

11226 SYNOPSIS

11227 #include <stdbool.h>

11228 DESCRIPTION

11229 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11230 conflict between the requirements described here and the ISO C standard is unintentional. This
11231 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11232 The **<stdbool.h>** header shall define the following macros:

11233 bool Expands to **_Bool**.

11234 true Expands to the integer constant 1.

11235 false Expands to the integer constant 0.

11236 __bool_true_false_are_defined

11237 Expands to the integer constant 1.

11238 An application may undefine and then possibly redefine the macros bool, true, and false.

11239 APPLICATION USAGE

11240 None.

11241 RATIONALE

11242 None.

11243 FUTURE DIRECTIONS

11244 The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature
11245 and may be withdrawn in a future version.

11246 SEE ALSO

11247 None.

11248 CHANGE HISTORY

11249 First released in Issue 6. Included for alignment with the ISO / IEC 9899: 1999 standard.

11250 **NAME**

11251 **stddef.h** — standard type definitions

11252 **SYNOPSIS**

11253 #include <stddef.h>

11254 **DESCRIPTION**

11255 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11256 conflict between the requirements described here and the ISO C standard is unintentional. This
11257 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11258 The <stddef.h> header shall define the following macros:

11259 **NULL** Null pointer constant.

11260 **offsetof**(*type*, *member-designator*)

11261 Integer constant expression of type **size_t**, the value of which is the offset in bytes
11262 to the structure member (*member-designator*), from the beginning of its structure
11263 (*type*).

11264 The <stddef.h> header shall define the following types:

11265 **ptrdiff_t** Signed integer type of the result of subtracting two pointers.

11266 **wchar_t** Integer type whose range of values can represent distinct wide-character codes for
11267 all members of the largest character set specified among the locales supported by
11268 the compilation environment: the null character has the code value 0 and each
11269 member of the portable character set has a code value equal to its value when used
11270 as the lone character in an integer character constant.

11271 **size_t** Unsigned integer type of the result of the *sizeof* operator.

11272 The implementation shall support one or more programming environments in which the widths
11273 of **ptrdiff_t**, **size_t**, and **wchar_t** are no greater than the width of type **long**. The names of these
11274 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

11275 **APPLICATION USAGE**

11276 None.

11277 **RATIONALE**

11278 None.

11279 **FUTURE DIRECTIONS**

11280 None.

11281 **SEE ALSO**

11282 <**wchar.h**>, <**sys/types.h**>, the System Interfaces volume of IEEE Std 1003.1-2001, *confstr()*, the
11283 Shell and Utilities volume of IEEE Std 1003.1-2001, *getconf*

11284 **CHANGE HISTORY**

11285 First released in Issue 4. Derived from the ANSI C standard.

11286 **NAME**

11287 stdint.h — integer types

11288 **SYNOPSIS**

11289 #include <stdint.h>

11290 **DESCRIPTION**

11291 **CX** Some of the functionality described on this reference page extends the ISO C standard.
 11292 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 11293 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 11294 symbols in this header.

11295 The <stdint.h> header shall declare sets of integer types having specified widths, and shall
 11296 define corresponding sets of macros. It shall also define macros that specify limits of integer
 11297 types corresponding to types defined in other standard headers.

11298 **Note:** The “width” of an integer type is the number of bits used to store its value in a pure binary
 11299 system; the actual type may use more bits than that (for example, a 28-bit type could be stored
 11300 in 32 bits of actual storage). An N -bit signed type has values in the range -2^{N-1} or $1-2^{N-1}$ to
 11301 $2^{N-1}-1$, while an N -bit unsigned type has values in the range 0 to 2^N-1 .

11302 Types are defined in the following categories:

- 11303 • Integer types having certain exact widths
- 11304 • Integer types having at least certain specified widths
- 11305 • Fastest integer types having at least certain specified widths
- 11306 • Integer types wide enough to hold pointers to objects
- 11307 • Integer types having greatest width

11308 (Some of these types may denote the same type.)

11309 Corresponding macros specify limits of the declared types and construct suitable constants.

11310 For each type described herein that the implementation provides, the <stdint.h> header shall
 11311 declare that **typedef** name and define the associated macros. Conversely, for each type described
 11312 herein that the implementation does not provide, the <stdint.h> header shall not declare that
 11313 **typedef** name, nor shall it define the associated macros. An implementation shall provide those
 11314 types described as required, but need not provide any of the others (described as optional).

11315 **Integer Types**

11316 When **typedef** names differing only in the absence or presence of the initial u are defined, they
 11317 shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999
 11318 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also
 11319 provide the other.

11320 In the following descriptions, the symbol N represents an unsigned decimal integer with no
 11321 leading zeros (for example, 8 or 24, but not 04 or 048).

- 11322 • Exact-width integer types

11323 The **typedef** name **int N _t** designates a signed integer type with width N , no padding bits,
 11324 and a two’s-complement representation. Thus, **int8_t** denotes a signed integer type with a
 11325 width of exactly 8 bits.

11326 The **typedef** name **uint N _t** designates an unsigned integer type with width N . Thus,
 11327 **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

11328	CX	The following types are required:
11329		int8_t
11330		int16_t
11331		int32_t
11332		uint8_t
11333		uint16_t
11334		uint32_t
11335		If an implementation provides integer types with width 64 that meet these requirements,
11336		then the following types are required:
11337		int64_t
11338		uint64_t
11339	CX	In particular, this will be the case if any of the following are true:
11340		— The implementation supports the <code>_POSIX_V6_ILP32_OFFBIG</code> programming
11341		environment and the application is being built in the <code>_POSIX_V6_ILP32_OFFBIG</code>
11342		programming environment (see the Shell and Utilities volume of IEEE Std 1003.1-2001,
11343		c99, Programming Environments).
11344		— The implementation supports the <code>_POSIX_V6_LP64_OFF64</code> programming environment
11345		and the application is being built in the <code>_POSIX_V6_LP64_OFF64</code> programming
11346		environment.
11347		— The implementation supports the <code>_POSIX_V6_LPBIG_OFFBIG</code> programming
11348		environment and the application is being built in the <code>_POSIX_V6_LPBIG_OFFBIG</code>
11349		programming environment.
11350		All other types of this form are optional.
11351		• Minimum-width integer types
11352		The typedef name int_leastN_t designates a signed integer type with a width of at least <i>N</i> ,
11353		such that no signed integer type with lesser size has at least the specified width. Thus,
11354		int_least32_t denotes a signed integer type with a width of at least 32 bits.
11355		The typedef name uint_leastN_t designates an unsigned integer type with a width of at least
11356		<i>N</i> , such that no unsigned integer type with lesser size has at least the specified width. Thus,
11357		uint_least16_t denotes an unsigned integer type with a width of at least 16 bits.
11358		The following types are required:
11359		int_least8_t
11360		int_least16_t
11361		int_least32_t
11362		int_least64_t
11363		uint_least8_t
11364		uint_least16_t
11365		uint_least32_t
11366		uint_least64_t
11367		All other types of this form are optional.
11368		• Fastest minimum-width integer types
11369		Each of the following types designates an integer type that is usually fastest to operate with
11370		among all integer types that have at least the specified width.

11371 The designated type is not guaranteed to be fastest for all purposes; if the implementation
 11372 has no clear grounds for choosing one type over another, it will simply pick some integer
 11373 type satisfying the signedness and width requirements.

11374 The **typedef** name **int_fastN_t** designates the fastest signed integer type with a width of at
 11375 least *N*. The **typedef** name **uint_fastN_t** designates the fastest unsigned integer type with a
 11376 width of at least *N*.

11377 The following types are required:

11378 **int_fast8_t**
 11379 **int_fast16_t**
 11380 **int_fast32_t**
 11381 **int_fast64_t**
 11382 **uint_fast8_t**
 11383 **uint_fast16_t**
 11384 **uint_fast32_t**
 11385 **uint_fast64_t**

11386 All other types of this form are optional.

- 11387 • Integer types capable of holding object pointers

11388 The following type designates a signed integer type with the property that any valid pointer
 11389 to **void** can be converted to this type, then converted back to a pointer to **void**, and the result
 11390 will compare equal to the original pointer:

11391 **intptr_t**

11392 The following type designates an unsigned integer type with the property that any valid
 11393 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
 11394 the result will compare equal to the original pointer:

11395 **uintptr_t**

11396 xsi On XSI-conformant systems, the **intptr_t** and **uintptr_t** types are required; otherwise, they
 11397 are optional.

- 11398 • Greatest-width integer types

11399 The following type designates a signed integer type capable of representing any value of any
 11400 signed integer type:

11401 **intmax_t**

11402 The following type designates an unsigned integer type capable of representing any value of
 11403 any unsigned integer type:

11404 **uintmax_t**

11405 These types are required.

11406 **Note:** Applications can test for optional types by using the corresponding limit macro from **Limits of**
 11407 **Specified-Width Integer Types** (on page 319).

Limits of Specified-Width Integer Types

The following macros specify the minimum and maximum limits of the types declared in the <stdint.h> header. Each macro name corresponds to a similar type name in **Integer Types** (on page 316).

Each instance of any defined macro shall be replaced by a constant expression suitable for use in **#if** preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign, except where stated to be exactly the given value.

- Limits of exact-width integer types

- Minimum values of exact-width signed integer types:

{INTN_MIN} Exactly $-(2^{N-1})$

- Maximum values of exact-width signed integer types:

{INTN_MAX} Exactly $2^{N-1} - 1$

- Maximum values of exact-width unsigned integer types:

{UINTN_MAX} Exactly $2^N - 1$

- Limits of minimum-width integer types

- Minimum values of minimum-width signed integer types:

{INT_LEASTN_MIN} $-(2^{N-1} - 1)$

- Maximum values of minimum-width signed integer types:

{INT_LEASTN_MAX} $2^{N-1} - 1$

- Maximum values of minimum-width unsigned integer types:

{UINT_LEASTN_MAX} $2^N - 1$

- Limits of fastest minimum-width integer types

- Minimum values of fastest minimum-width signed integer types:

{INT_FASTN_MIN} $-(2^{N-1} - 1)$

- Maximum values of fastest minimum-width signed integer types:

{INT_FASTN_MAX} $2^{N-1} - 1$

- Maximum values of fastest minimum-width unsigned integer types:

{UINT_FASTN_MAX} $2^N - 1$

- Limits of integer types capable of holding object pointers

- Minimum value of pointer-holding signed integer type:

{INTPTR_MIN} $-(2^{15} - 1)$

- Maximum value of pointer-holding signed integer type:

{INTPTR_MAX} $2^{15} - 1$

- Maximum value of pointer-holding unsigned integer type:

11445 {UINTPTR_MAX} $2^{16} - 1$

11446 • Limits of greatest-width integer types

11447 — Minimum value of greatest-width signed integer type:

11448 {INTMAX_MIN} $-(2^{63} - 1)$

11449 — Maximum value of greatest-width signed integer type:

11450 {INTMAX_MAX} $2^{63} - 1$

11451 — Maximum value of greatest-width unsigned integer type:

11452 {UINTMAX_MAX} $2^{64} - 1$

11453 Limits of Other Integer Types

11454 The following macros specify the minimum and maximum limits of integer types corresponding
11455 to types defined in other standard headers.

11456 Each instance of these macros shall be replaced by a constant expression suitable for use in **#if**
11457 preprocessing directives, and this expression shall have the same type as would an expression
11458 that is an object of the corresponding type converted according to the integer promotions. Its
11459 implementation-defined value shall be equal to or greater in magnitude (absolute value) than
11460 the corresponding value given below, with the same sign.

11461 • Limits of **ptrdiff_t**:

11462 {PTRDIFF_MIN} $-65\,535$

11463 {PTRDIFF_MAX} $+65\,535$

11464 • Limits of **sig_atomic_t**:

11465 {SIG_ATOMIC_MIN} See below.

11466 {SIG_ATOMIC_MAX} See below.

11467 • Limit of **size_t**:

11468 {SIZE_MAX} $65\,535$

11469 • Limits of **wchar_t**:

11470 {WCHAR_MIN} See below.

11471 {WCHAR_MAX} See below.

11472 • Limits of **wint_t**:

11473 {WINT_MIN} See below.

11474 {WINT_MAX} See below.

11475 If **sig_atomic_t** (see the <signal.h> header) is defined as a signed integer type, the value of
11476 {SIG_ATOMIC_MIN} shall be no greater than -127 and the value of {SIG_ATOMIC_MAX} shall
11477 be no less than 127 ; otherwise, **sig_atomic_t** shall be defined as an unsigned integer type, and the
11478 value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no less
11479 than 255 .

11480 If **wchar_t** (see the <stddef.h> header) is defined as a signed integer type, the value of
11481 {WCHAR_MIN} shall be no greater than -127 and the value of {WCHAR_MAX} shall be no less
11482 than 127 ; otherwise, **wchar_t** shall be defined as an unsigned integer type, and the value of
11483 {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255 .

11484 If **wint_t** (see the <wchar.h> header) is defined as a signed integer type, the value of
 11485 {WINT_MIN} shall be no greater than −32 767 and the value of {WINT_MAX} shall be no less
 11486 than 32 767; otherwise, **wint_t** shall be defined as an unsigned integer type, and the value of
 11487 {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65 535.

11488 **Macros for Integer Constant Expressions**

11489 The following macros expand to integer constant expressions suitable for initializing objects that
 11490 have integer types corresponding to types defined in the <stdint.h> header. Each macro name
 11491 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*
 11492 *integer types*.

11493 Each invocation of one of these macros shall expand to an integer constant expression suitable
 11494 for use in **#if** preprocessing directives. The type of the expression shall have the same type as
 11495 would an expression that is an object of the corresponding type converted according to the
 11496 integer promotions. The value of the expression shall be that of the argument.

11497 The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant
 11498 with a value that does not exceed the limits for the corresponding type.

11499 • Macros for minimum-width integer constant expressions

11500 The macro **INTN_C(value)** shall expand to an integer constant expression corresponding to
 11501 the type **int_leastN_t**. The macro **UINTN_C(value)** shall expand to an integer constant
 11502 expression corresponding to the type **uint_leastN_t**. For example, if **uint_least64_t** is a name
 11503 for the type **unsigned long long**, then **UINT64_C(0x123)** might expand to the integer
 11504 constant 0x123ULL.

11505 • Macros for greatest-width integer constant expressions

11506 The following macro expands to an integer constant expression having the value specified by
 11507 its argument and the type **intmax_t**:

11508 **INTMAX_C(value)**

11509 The following macro expands to an integer constant expression having the value specified by
 11510 its argument and the type **uintmax_t**:

11511 **UINTMAX_C(value)**

11512 **APPLICATION USAGE**

11513 None.

11514 **RATIONALE**

11515 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in
 11516 freestanding environments, which might not support the formatted I/O functions. In some
 11517 environments, if the formatted conversion support is not wanted, using this header instead of
 11518 the <inttypes.h> header avoids defining such a large number of macros.

11519 As a consequence of adding **int8_t**, the following are true:

- 11520 • A byte is exactly 8 bits.
- 11521 • {CHAR_BIT} has the value 8, {SCHAR_MAX} has the value 127, {SCHAR_MIN} has the
 11522 value −127 or −128, and {UCHAR_MAX} has the value 255.

11523 **FUTURE DIRECTIONS**

11524 **typedef** names beginning with **int** or **uint** and ending with **_t** may be added to the types defined
 11525 in the <stdint.h> header. Macro names beginning with **INT** or **UINT** and ending with **_MAX**,
 11526 **_MIN**, or **_C** may be added to the macros defined in the <stdint.h> header.

11527 **SEE ALSO**11528 **<inttypes.h>**, **<signal.h>**, **<stddef.h>**, **<wchar.h>**11529 **CHANGE HISTORY**

11530 First released in Issue 6. Included for alignment with the ISO / IEC 9899: 1999 standard.

11531 ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated. |

11532 NAME

11533 stdio.h — standard buffered input/output

11534 SYNOPSIS

11535 #include <stdio.h>

11536 DESCRIPTION

11537 CX Some of the functionality described on this reference page extends the ISO C standard.
 11538 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 11539 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 11540 symbols in this header.

11541 The <stdio.h> header shall define the following macros as positive integer constant expressions:

11542 BUFSIZ Size of <stdio.h> buffers.

11543 _IIOFBF Input/output fully buffered.

11544 _IIOBFB Input/output line buffered.

11545 _IIONBF Input/output unbuffered.

11546 CX L_ctermid Maximum size of character array to hold *ctermid()* output.

11547 L_tmpnam Maximum size of character array to hold *tmpnam()* output.

11548 SEEK_CUR Seek relative to current position.

11549 SEEK_END Seek relative to end-of-file.

11550 SEEK_SET Seek relative to start-of-file.

11551 The following macros shall be defined as positive integer constant expressions which denote
 11552 implementation limits:

11553 {FILENAME_MAX} Maximum size in bytes of the longest filename string that the
 11554 implementation guarantees can be opened.

11555 {FOPEN_MAX} Number of streams which the implementation guarantees can be open
 11556 simultaneously. The value is at least eight.

11557 {TMP_MAX} Minimum number of unique filenames generated by *tmpnam()*.
 11558 Maximum number of times an application can call *tmpnam()* reliably. The
 11559 XSI value of {TMP_MAX} is at least 25. On XSI-conformant systems, the
 11560 value of {TMP_MAX} is at least 10 000.

11561 The following macro name shall be defined as a negative integer constant expression:

11562 EOF End-of-file return value.

11563 The following macro name shall be defined as a null pointer constant:

11564 NULL Null pointer.

11565 The following macro name shall be defined as a string constant:

11566 XSI P_tmpdir Default directory prefix for *tmpnam()*.

11567 The following shall be defined as expressions of type “pointer to **FILE**” that point to the **FILE**
 11568 objects associated, respectively, with the standard error, input, and output streams:

11569 stderr Standard error output stream.

11570 stdin Standard input stream.

11571	<i>stdout</i>	Standard output stream.
11572	The following data types shall be defined through typedef :	
11573	FILE	A structure containing information about a file.
11574	fpos_t	A non-array type containing all information needed to specify uniquely
11575		every position within a file.
11576 XSI	va_list	As described in <stdarg.h>.
11577	size_t	As described in <stddef.h>.
11578	The following shall be declared as functions and may also be defined as macros. Function	
11579	prototypes shall be provided.	
11580	void	clearerr(FILE *);
11581 CX	char	*ctermid(char *);
11582	int	fclose(FILE *);
11583 CX	FILE	*fdopen(int, const char *);
11584	int	feof(FILE *);
11585	int	ferror(FILE *);
11586	int	fflush(FILE *);
11587	int	fgetc(FILE *);
11588	int	fgetpos(FILE *restrict, fpos_t *restrict);
11589	char	*fgets(char *restrict, int, FILE *restrict);
11590 CX	int	fileno(FILE *);
11591 TSF	void	flockfile(FILE *);
11592	FILE	*fopen(const char *restrict, const char *restrict);
11593	int	fprintf(FILE *restrict, const char *restrict, ...);
11594	int	fputc(int, FILE *);
11595	int	fputs(const char *restrict, FILE *restrict);
11596	size_t	fread(void *restrict, size_t, size_t, FILE *restrict);
11597	FILE	*freopen(const char *restrict, const char *restrict,
11598		FILE *restrict);
11599	int	fscanf(FILE *restrict, const char *restrict, ...);
11600	int	fseek(FILE *, long, int);
11601 CX	int	fseeko(FILE *, off_t, int);
11602	int	fsetpos(FILE *, const fpos_t *);
11603	long	ftell(FILE *);
11604 CX	off_t	ftello(FILE *);
11605 TSF	int	ftrylockfile(FILE *);
11606	void	funlockfile(FILE *);
11607	size_t	fwrite(const void *restrict, size_t, size_t, FILE *restrict);
11608	int	getc(FILE *);
11609	int	getchar(void);
11610 TSF	int	getc_unlocked(FILE *);
11611	int	getchar_unlocked(void);
11612	char	*gets(char *);
11613 CX	int	pclose(FILE *);
11614	void	perror(const char *);
11615 CX	FILE	*popen(const char *, const char *);
11616	int	printf(const char *restrict, ...);
11617	int	putc(int, FILE *);
11618	int	putchar(int);
11619 TSF		

```

11620     int      putc_unlocked(int, FILE *);
11621     int      putchar_unlocked(int);
11622     int      puts(const char *);
11623     int      remove(const char *);
11624     int      rename(const char *, const char *);
11625     void     rewind(FILE *);
11626     int      scanf(const char *restrict, ...);
11627     void     setbuf(FILE *restrict, char *restrict);
11628     int      setvbuf(FILE *restrict, char *restrict, int, size_t);
11629     int      snprintf(char *restrict, size_t, const char *restrict, ...);
11630     int      sprintf(char *restrict, const char *restrict, ...);
11631     int      sscanf(const char *restrict, const char *restrict, int ...);
11632 XSI     char  *tempnam(const char *, const char *);
11633     FILE     *tmpfile(void);
11634     char     *tmpnam(char *);
11635     int      ungetc(int, FILE *);
11636     int      vfprintf(FILE *restrict, const char *restrict, va_list);
11637     int      vfprintf(FILE *restrict, const char *restrict, va_list);
11638     int      vprintf(const char *restrict, va_list);
11639     int      vscanf(const char *restrict, va_list);
11640     int      vsnprintf(char *restrict, size_t, const char *restrict, va_list);
11641     int      vsprintf(char *restrict, const char *restrict, va_list);
11642     int      vsscanf(const char *restrict, const char *restrict, va_list arg);

```

11643 XSI Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.

11644 APPLICATION USAGE

11645 None.

11646 RATIONALE

11647 None.

11648 FUTURE DIRECTIONS

11649 None.

11650 SEE ALSO

11651 <stdarg.h>, <stddef.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001,
11652 *clearerr()*, *ctermid()*, *fclose()*, *fdopen()*, *fgetc()*, *fgetpos()*, *ferror()*, *feof()*, *fflush()*, *fgets()*, *fileno()*,
11653 *flockfile()*, *fopen()*, *fputc()*, *fputs()*, *fread()*, *freopen()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*,
11654 *getc_unlocked()*, *getwchar()*, *getchar()*, *getopt()*, *gets()*, *pclose()*, *perror()*, *popen()*, *printf()*, *putc()*,
11655 *putchar()*, *puts()*, *putwchar()*, *remove()*, *rename()*, *rewind()*, *scanf()*, *setbuf()*, *setvbuf()*, *sscanf()*,
11656 *stdin*, *system()*, *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vfprintf()*, *vscanf()*, *vprintf()*, *vsscanf()*

11657 CHANGE HISTORY

11658 First released in Issue 1. Derived from Issue 1 of the SVID.

11659 Issue 5

11660 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11661 Large File System extensions are added.

11662 The constant *L_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as
11663 extensions and LEGACY.

11664 The *cuserid()* and *getopt()* functions are marked LEGACY.

11665 **Issue 6**

11666 The constant `L_cuserid` and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed
11667 as they were previously marked LEGACY.

11668 The *cuserid()*, *getopt()*, and *getw()* functions are removed as they were previously marked
11669 LEGACY.

11670 Several functions are marked as part of the Thread-Safe Functions option.

11671 This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the
11672 description of the **fpos_t** type is now explicitly updated to exclude array types.

11673 Extensions beyond the ISO C standard are marked.

11674 **NAME**

11675 stdlib.h — standard library definitions

11676 **SYNOPSIS**

11677 #include <stdlib.h>

11678 **DESCRIPTION**

11679 CX Some of the functionality described on this reference page extends the ISO C standard.
11680 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
11681 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
11682 symbols in this header.

11683 The <stdlib.h> header shall define the following macros:

11684 EXIT_FAILURE Unsuccessful termination for *exit()*; evaluates to a non-zero value.

11685 EXIT_SUCCESS Successful termination for *exit()*; evaluates to 0.

11686 NULL Null pointer.

11687 {RAND_MAX} Maximum value returned by *rand()*; at least 32 767.

11688 {MB_CUR_MAX} Integer expression whose value is the maximum number of bytes in a
11689 character specified by the current locale.

11690 The following data types shall be defined through **typedef**:

11691 **div_t** Structure type returned by the *div()* function.

11692 **ldiv_t** Structure type returned by the *ldiv()* function.

11693 **lldiv_t** Structure type returned by the *lldiv()* function.

11694 **size_t** As described in <stddef.h>.

11695 **wchar_t** As described in <stddef.h>.

11696 In addition, the following symbolic names and macros shall be defined as in <sys/wait.h>, for
11697 use in decoding the return value from *system()*:

11698 XSI WNOHANG
11699 WUNTRACED
11700 WEXITSTATUS
11701 WIFEXITED
11702 WIFSIGNALED
11703 WIFSTOPPED
11704 WSTOPSIG
11705 WTERMSIG
11706

11707 The following shall be declared as functions and may also be defined as macros. Function
11708 prototypes shall be provided.

11709 void _Exit(int);
11710 XSI long a64l(const char *);
11711 void abort(void);
11712 int abs(int);
11713 int atexit(void (*)(void));
11714 double atof(const char *);
11715 int atoi(const char *);
11716 long atol(const char *);

```

11717 long long      atoll(const char *);
11718 void          *bsearch(const void *, const void *, size_t, size_t,
11719                      int (*)(const void *, const void *));
11720 void          *calloc(size_t, size_t);
11721 div_t         div(int, int);
11722 XSI double     drand48(void);
11723 char          *ecvt(double, int, int *restrict, int *restrict); (LEGACY)
11724 double        erand48(unsigned short[3]);
11725 void          exit(int);
11726 XSI char       *fcvt(double, int, int *restrict, int *restrict); (LEGACY)
11727 void          free(void *);
11728 XSI char       *gcvt(double, int, char *); (LEGACY)
11729 char          *getenv(const char *);
11730 XSI int        getsuopt(char **, char *const *, char **);
11731 int           grantpt(int);
11732 char          *initstate(unsigned, char *, size_t);
11733 long          jrand48(unsigned short[3]);
11734 char          *l64a(long);
11735 long          labs(long);
11736 XSI void       lcong48(unsigned short[7]);
11737 ldiv_t        ldiv(long, long);
11738 long long     llabs(long long);
11739 lldiv_t       lldiv(long long, long long);
11740 XSI long       lrand48(void);
11741 void          *malloc(size_t);
11742 int           mblen(const char *, size_t);
11743 size_t        mbstowcs(wchar_t *restrict, const char *restrict, size_t);
11744 int           mbtowc(wchar_t *restrict, const char *restrict, size_t);
11745 XSI char       *mktemp(char *); (LEGACY)
11746 int           mkstemp(char *);
11747 long          mrand48(void);
11748 long          nrand48(unsigned short[3]);
11749 ADV int        posix_memalign(void **, size_t, size_t);
11750 XSI int        posix_openpt(int);
11751 char          *ptsname(int);
11752 int           putenv(char *);
11753 void          qsort(void *, size_t, size_t, int (*)(const void *,
11754                      const void *));
11755 int           rand(void);
11756 TSF int        rand_r(unsigned *);
11757 XSI long       random(void);
11758 void          *realloc(void *, size_t);
11759 XSI char       *realpath(const char *restrict, char *restrict);
11760 unsigned short seed48(unsigned short[3]);
11761 CX int         setenv(const char *, const char *, int);
11762 XSI void       setkey(const char *);
11763 char          *setstate(const char *);
11764 void          srand(unsigned);
11765 XSI void       srand48(long);
11766 void          srandom(unsigned);
11767 double        strtod(const char *restrict, char **restrict);
11768 float         strtod(const char *restrict, char **restrict);

```

```

11769      long          strtol(const char *restrict, char **restrict, int);
11770      long double    strtold(const char *restrict, char **restrict);
11771      long long      strtoll(const char *restrict, char **restrict, int);
11772      unsigned long   strtoul(const char *restrict, char **restrict, int);
11773      unsigned long   long
11774                        strtoull(const char *restrict, char **restrict, int);
11775      int             system(const char *);
11776 XSI      int         unlockpt(int);
11777 CX      int         unsetenv(const char *);
11778      size_t          wcstombs(char *restrict, const wchar_t *restrict, size_t);
11779      int             wctomb(char *, wchar_t);

11780 XSI      Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>,
11781            <limits.h>, <math.h>, and <sys/wait.h>.

```

11782 APPLICATION USAGE

11783 None.

11784 RATIONALE

11785 None.

11786 FUTURE DIRECTIONS

11787 None.

11788 SEE ALSO

11789 <limits.h>, <math.h>, <stddef.h>, <sys/types.h>, <sys/wait.h>, the System Interfaces volume of
11790 IEEE Std 1003.1-2001, *_Exit()*, *a64l()*, *abort()*, *abs()*, *atexit()*, *atof()*, *atoi()*, *atoll()*, *bsearch()*,
11791 *calloc()*, *div()*, *drand48()*, *erand48()*, *exit()*, *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*,
11792 *jrand48()*, *l64a()*, *labs()*, *lcong48()*, *ldiv()*, *llabs()*, *lldiv()*, *lrand48()*, *malloc()*, *mblen()*, *mbstowcs()*,
11793 *mbtowc()*, *mkstemp()*, *mrand48()*, *nrand48()*, *posix_memalign()*, *ptsname()*, *putenv()*, *qsort()*,
11794 *rand()*, *realloc()*, *realpath()*, *setstate()*, *srand()*, *srand48()*, *srandom()*, *strtod()*, *strtof()*, *strtol()*,
11795 *strtold()*, *strtoll()*, *strtoul()*, *strtoull()*, *unlockpt()*, *wcstombs()*, *wctomb()*

11796 CHANGE HISTORY

11797 First released in Issue 3.

11798 Issue 5

11799 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11800 The *ttyslot()* and *valloc()* functions are marked LEGACY.

11801 The type of the third argument to *initstate()* is changed from **int** to **size_t**. The type of the return
11802 value from *setstate()* is changed from **char** to **char ***, and the type of the first argument is
11803 changed from **char *** to **const char ***.

11804 Issue 6

11805 The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath()* to be
11806 consistent with the reference page.

11807 The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv()* to be
11808 consistent with the reference page.

11809 The *rand_r()* function is marked as part of the Thread-Safe Functions option.

11810 Function prototypes for *setenv()* and *unsetenv()* are added.

11811 The *posix_memalign()* function is added for alignment with IEEE Std 1003.1d-1999.

11812 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

- 11813 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.
- 11814 The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY.
- 11815 Extensions beyond the ISO C standard are marked.

11816 **NAME**

11817 string.h — string operations

11818 **SYNOPSIS**

11819 #include <string.h>

11820 **DESCRIPTION**

11821 CX Some of the functionality described on this reference page extends the ISO C standard.
 11822 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 11823 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 11824 symbols in this header.

11825 The <string.h> header shall define the following:

11826 NULL Null pointer constant.

11827 **size_t** As described in <stddef.h>.

11828 The following shall be declared as functions and may also be defined as macros. Function
 11829 prototypes shall be provided.

```

11830 XSI void *memcpy(void *restrict, const void *restrict, int, size_t);
11831 void *memchr(const void *, int, size_t);
11832 int memcmp(const void *, const void *, size_t);
11833 void *memcpy(void *restrict, const void *restrict, size_t);
11834 void *memmove(void *, const void *, size_t);
11835 void *memset(void *, int, size_t);
11836 char *strcat(char *restrict, const char *restrict);
11837 char *strchr(const char *, int);
11838 int strcmp(const char *, const char *);
11839 int strcoll(const char *, const char *);
11840 char *strcpy(char *restrict, const char *restrict);
11841 size_t strcspn(const char *, const char *);
11842 XSI char *strdup(const char *);
11843 char *strerror(int);
11844 TSF int *strerror_r(int, char *, size_t);
11845 size_t strlen(const char *);
11846 char *strncat(char *restrict, const char *restrict, size_t);
11847 int strncmp(const char *, const char *, size_t);
11848 char *strncpy(char *restrict, const char *restrict, size_t);
11849 char *strpbrk(const char *, const char *);
11850 char *strrchr(const char *, int);
11851 size_t strspn(const char *, const char *);
11852 char *strstr(const char *, const char *);
11853 char *strtok(char *restrict, const char *restrict);
11854 TSF char *strtok_r(char *, const char *, char **);
11855 size_t strxfrm(char *restrict, const char *restrict, size_t);
    
```

11856 XSI Inclusion of the <string.h> header may also make visible all symbols from <stddef.h>.

11857 **APPLICATION USAGE**

11858 None.

11859 **RATIONALE**

11860 None.

11861 **FUTURE DIRECTIONS**

11862 None.

11863 **SEE ALSO**

11864 <stddef.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *memcpy()*,
11865 *memchr()*, *memcmp()*, *memcpy()*, *memmove()*, *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*,
11866 *strcpy()*, *strcspn()*, *strdup()*, *strerror()*, *strlen()*, *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*,
11867 *strspn()*, *strstr()*, *strtok()*, *strxfrm()*

11868 **CHANGE HISTORY**

11869 First released in Issue 1. Derived from Issue 1 of the SVID.

11870 **Issue 5**

11871 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11872 **Issue 6**11873 The *strtok_r()* function is marked as part of the Thread-Safe Functions option.

11874 This reference page is updated to align with the ISO / IEC 9899:1999 standard.

11875 The *strerror_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

11876 **NAME**

11877 strings.h — string operations

11878 **SYNOPSIS**

11879 xSI #include <strings.h>

11880

11881 **DESCRIPTION**

11882 The following shall be declared as functions and may also be defined as macros. Function
 11883 prototypes shall be provided.

11884 int bcmp(const void *, const void *, size_t); (**LEGACY**)11885 void bcopy(const void *, void *, size_t); (**LEGACY**)11886 void bzero(void *, size_t); (**LEGACY**)

11887 int ffs(int);

11888 char *index(const char *, int); (**LEGACY**)11889 char *rindex(const char *, int); (**LEGACY**)

11890 int strcasecmp(const char *, const char *);

11891 int strncasecmp(const char *, const char *, size_t);

11892 The **size_t** type shall be defined through **typedef** as described in <stddef.h>.11893 **APPLICATION USAGE**

11894 None.

11895 **RATIONALE**

11896 None.

11897 **FUTURE DIRECTIONS**

11898 None.

11899 **SEE ALSO**

11900 <stddef.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *ffs()*, *strcasecmp()*,
 11901 *strncasecmp()*

11902 **CHANGE HISTORY**

11903 First released in Issue 4, Version 2.

11904 **Issue 6**

11905 The Open Group Corrigendum U021/2 is applied, correcting the prototype for *index()* to be
 11906 consistent with the reference page.

11907 The *bcmp()*, *bcopy()*, *bzero()*, *index()*, and *rindex()* functions are marked LEGACY.

11908 **NAME**11909 stropts.h — STREAMS interface (**STREAMS**)11910 **SYNOPSIS**

11911 xSR #include <stropts.h>

11912

11913 **DESCRIPTION**11914 The <stropts.h> header shall define the **bandinfo** structure that includes at least the following
11915 members:

11916	unsigned char	bi_pri	Priority band.
11917	int	bi_flag	Flushing type.

11918 The <stropts.h> header shall define the **strpeek** structure that includes at least the following
11919 members:

11920	struct strbuf	ctlbuf	The control portion of the message.
11921	struct strbuf	databuf	The data portion of the message.
11922	t_uscalar_t	flags	RS_HIPRI or 0.

11923 The <stropts.h> header shall define the **strbuf** structure that includes at least the following
11924 members:

11925	int	maxlen	Maximum buffer length.
11926	int	len	Length of data.
11927	char	*buf	Pointer to buffer.

11928 The <stropts.h> header shall define the **strfdinsert** structure that includes at least the following
11929 members:

11930	struct strbuf	ctlbuf	The control portion of the message.
11931	struct strbuf	databuf	The data portion of the message.
11932	t_uscalar_t	flags	RS_HIPRI or 0.
11933	int	fildes	File descriptor of the other STREAM.
11934	int	offset	Relative location of the stored value.

11935 The <stropts.h> header shall define the **strioctl** structure that includes at least the following
11936 members:

11937	int	ic_cmd	ioctl() command.
11938	int	ic_timeout	Timeout for response.
11939	int	ic_len	Length of data.
11940	char	*ic_dp	Pointer to buffer.

11941 The <stropts.h> header shall define the **strrecvfd** structure that includes at least the following
11942 members:

11943	int	fda	Received file descriptor.
11944	uid_t	uid	UID of sender.
11945	gid_t	gid	GID of sender.

11946 The **uid_t** and **gid_t** types shall be defined through **typedef** as described in <sys/types.h>.11947 The <stropts.h> header shall define the **t_scalar_t** and **t_uscalar_t** types, respectively, as signed
11948 and unsigned opaque types of equal length of at least 32 bits.11949 The <stropts.h> header shall define the **str_list** structure that includes at least the following
11950 members:

```

11951      int                sl_nmods      Number of STREAMS module names.
11952      struct str_mlist  *sl_modlist   STREAMS module names.

11953      The <stropts.h> header shall define the str_mlist structure that includes at least the following
11954      member:

11955      char  l_name[FMNAMESZ+1]  A STREAMS module name.

11956      At least the following macros shall be defined for use as the request argument to ioctl():

11957      I_PUSH          Push a STREAMS module.
11958      I_POP           Pop a STREAMS module.
11959      I_LOOK          Get the top module name.
11960      I_FLUSH         Flush a STREAM.
11961      I_FLUSHBAND     Flush one band of a STREAM.
11962      I_SETSIG        Ask for notification signals.
11963      I_GETSIG        Retrieve current notification signals.
11964      I_FIND          Look for a STREAMS module.
11965      I_PEEK          Peek at the top message on a STREAM.
11966      I_SRDOPT        Set the read mode.
11967      I_GRDOPT        Get the read mode.
11968      I_NREAD         Size the top message.
11969      I_FDINSERT      Send implementation-defined information about another STREAM.
11970      I_STR           Send a STREAMS ioctl().
11971      I_SWROPT        Set the write mode.
11972      I_GWROPT        Get the write mode.
11973      I_SENDFD        Pass a file descriptor through a STREAMS pipe.
11974      I_RECVFD        Get a file descriptor sent via I_SENDFD.
11975      I_LIST          Get all the module names on a STREAM.
11976      I_ATMARK        Is the top message “marked”?
11977      I_CKBAND        See if any messages exist in a band.
11978      I_GETBAND       Get the band of the top message on a STREAM.
11979      I_CANPUT        Is a band writable?
11980      I_SETCLTIME     Set close time delay.
11981      I_GETCLTIME     Get close time delay.
11982      I_LINK          Connect two STREAMs.
11983      I_UNLINK        Disconnect two STREAMs.
11984      I_PLINK         Persistently connect two STREAMs.
11985      I_PUNLINK       Dismantle a persistent STREAMS link.

```

11986	At least the following macros shall be defined for use with I_LOOK:	
11987	FMNAMESZ	The minimum size in bytes of the buffer referred to by the <i>arg</i> argument.
11988	At least the following macros shall be defined for use with I_FLUSH:	
11989	FLUSHR	Flush read queues.
11990	FLUSHW	Flush write queues.
11991	FLUSHRW	Flush read and write queues.
11992	At least the following macros shall be defined for use with I_SETSIG:	
11993	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a
11994		STREAM head read queue.
11995	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM
11996		head read queue.
11997	S_INPUT	A message, other than a high-priority message, has arrived at the head of a
11998		STREAM head read queue.
11999	S_HIPRI	A high-priority message is present on a STREAM head read queue.
12000	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM
12001		head is no longer full. This notifies the process that there is room on the queue
12002		for sending (or writing) normal data downstream.
12003	S_WRNORM	Equivalent to S_OUTPUT.
12004	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is
12005		no longer full.
12006	S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the
12007		front of the STREAM head read queue.
12008	S_ERROR	Notification of an error condition reaches the STREAM head.
12009	S_HANGUP	Notification of a hangup reaches the STREAM head.
12010	S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of
12011		SIGPOLL when a priority message reaches the front of the STREAM head read
12012		queue.
12013	At least the following macros shall be defined for use with I_PEEK:	
12014	RS_HIPRI	Only look for high-priority messages.
12015	At least the following macros shall be defined for use with I_SRDOPT:	
12016	RNORM	Byte-STREAM mode, the default.
12017	RMSGD	Message-discard mode.
12018	RMSGN	Message-non-discard mode.
12019	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the
12020		front of the STREAM head read queue.
12021	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
12022	RPROTDIS	Discard the control part of a message, delivering any data part, when a
12023		process issues a <i>read()</i> .

12024 At least the following macros shall be defined for use with I_SWOPT:

12025 SNDZERO Send a zero-length message downstream when a *write()* of 0 bytes occurs.

12026 At least the following macros shall be defined for use with I_ATMARK:

12027 ANYMARK Check if the message is marked.

12028 LASTMARK Check if the message is the last one marked on the queue.

12029 At least the following macros shall be defined for use with I_UNLINK:

12030 MUXID_ALL Unlink all STREAMs linked to the STREAM associated with *fildev*.

12031 The following macros shall be defined for *getmsg()*, *getpmsg()*, *putmsg()*, and *putpmsg()*:

12032 MSG_ANY Receive any message.

12033 MSG_BAND Receive message from specified band.

12034 MSG_HIPRI Send/receive high-priority message.

12035 MORECTL More control information is left in message.

12036 MOREDATA More data is left in message.

12037 The <stropts.h> header may make visible all of the symbols from <unistd.h>.

12038 The following shall be declared as functions and may also be defined as macros. Function

12039 prototypes shall be provided.

12040 int isastream(int);

12041 int getmsg(int, struct strbuf *restrict, struct strbuf *restrict,

12042 int *restrict);

12043 int getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,

12044 int *restrict, int *restrict);

12045 int ioctl(int, int, ...);

12046 int putmsg(int, const struct strbuf *, const struct strbuf *, int);

12047 int putpmsg(int, const struct strbuf *, const struct strbuf *, int,

12048 int);

12049 int fattach(int, const char *);

12050 int fdetach(const char *);

12051 **APPLICATION USAGE**

12052 None.

12053 **RATIONALE**

12054 None.

12055 **FUTURE DIRECTIONS**

12056 None.

12057 **SEE ALSO**

12058 <sys/types.h>, <unistd.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *close()*, *fcntl()*,

12059 *getmsg()*, *ioctl()*, *open()*, *pipe()*, *read()*, *poll()*, *putmsg()*, *signal()*, *write()*

12060 **CHANGE HISTORY**

12061 First released in Issue 4, Version 2.

12062 **Issue 5**

12063 The *flags* members of the **strpeek** and **strfdinsert** structures are changed from **type long** to
12064 **t_uscalar_t**.

12065 **Issue 6**

12066 This header is marked as part of the XSI STREAMS Option Group.

12067 The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.

12068 **NAME**

12069 sys/ipc.h — XSI interprocess communication access structure

12070 **SYNOPSIS**

12071 xsi #include <sys/ipc.h>

12072

12073 **DESCRIPTION**

12074 The <sys/ipc.h> header is used by three mechanisms for XSI interprocess communication (IPC):
 12075 messages, semaphores, and shared memory. All use a common structure type, **ipc_perm**, to pass
 12076 information used in determining permission to perform an IPC operation.

12077 The **ipc_perm** structure shall contain the following members:

12078	uid_t	uid	Owner's user ID.
12079	gid_t	gid	Owner's group ID.
12080	uid_t	cuid	Creator's user ID.
12081	gid_t	cgid	Creator's group ID.
12082	mode_t	mode	Read/write permission.

12083 The **uid_t**, **gid_t**, **mode_t**, and **key_t** types shall be defined as described in <sys/types.h>.

12084 Definitions shall be provided for the following constants:

12085 Mode bits:

12086	IPC_CREAT	Create entry if key does not exist.
12087	IPC_EXCL	Fail if key exists.
12088	IPC_NOWAIT	Error if request must wait.

12089 Keys:

12090	IPC_PRIVATE	Private key.

12091 Control commands:

12092	IPC_RMID	Remove identifier.
12093	IPC_SET	Set options.
12094	IPC_STAT	Get options.

12095 The following shall be declared as a function and may also be defined as a macro. A function
 12096 prototype shall be provided.

12097 key_t ftok(const char *, int);

12098 **APPLICATION USAGE**

12099 None.

12100 **RATIONALE**

12101 None.

12102 **FUTURE DIRECTIONS**

12103 None.

12104 **SEE ALSO**

12105 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *ftok()*

12106 **CHANGE HISTORY**

12107 First released in Issue 2. Derived from System V Release 2.0.

12108 **NAME**

12109 sys/mman.h — memory management declarations

12110 **SYNOPSIS**

12111 #include <sys/mman.h>

12112 **DESCRIPTION**

12113 The <sys/mman.h> header shall be supported if the implementation supports at least one of the
12114 following options:

- 12115 MF • The Memory Mapped Files option
- 12116 SHM • The Shared Memory Objects option
- 12117 ML • The Process Memory Locking option
- 12118 MPR • The Memory Protection option
- 12119 TYM • The Typed Memory Objects option
- 12120 SIO • The Synchronized Input and Output option
- 12121 ADV • The Advisory Information option
- 12122 TYM • The Typed Memory Objects option

12123 MC2 If one or more of the Advisory Information, Memory Mapped Files, or Shared Memory Objects
12124 options are supported, the following protection options shall be defined:

- 12125 MC2 PROT_READ Page can be read.
- 12126 MC2 PROT_WRITE Page can be written.
- 12127 MC2 PROT_EXEC Page can be executed.
- 12128 MC2 PROT_NONE Page cannot be accessed.

12129 The following *flag* options shall be defined:

- 12130 MF|SHM MAP_SHARED Share changes.
- 12131 MF|SHM MAP_PRIVATE Changes are private.
- 12132 MF|SHM MAP_FIXED Interpret *addr* exactly.

12133 The following flags shall be defined for *msync()*:

- 12134 MF|SIO MS_ASYNC Perform asynchronous writes.
- 12135 MF|SIO MS_SYNC Perform synchronous writes.
- 12136 MF|SIO MS_INVALIDATE Invalidate mappings.

12137 ML The following symbolic constants shall be defined for the *mlockall()* function:

- 12138 ML MCL_CURRENT Lock currently mapped pages.
- 12139 ML MCL_FUTURE Lock pages that become mapped.

12140 MF|SHM The symbolic constant MAP_FAILED shall be defined to indicate a failure from the *mmap()*
12141 function.

12142 MC1 If the Advisory Information and either the Memory Mapped Files or Shared Memory Objects
12143 options are supported, values for *advice* used by *posix_madvise()* shall be defined as follows:

- 12144 POSIX_MADV_NORMAL
- 12145 The application has no advice to give on its behavior with respect to the specified range. It

12146		is the default characteristic if no advice is given for a range of memory.
12147		POSIX_MADV_SEQUENTIAL
12148		The application expects to access the specified range sequentially from lower addresses to
12149		higher addresses.
12150		POSIX_MADV_RANDOM
12151		The application expects to access the specified range in a random order.
12152		POSIX_MADV_WILLNEED
12153		The application expects to access the specified range in the near future.
12154		POSIX_MADV_DONTNEED
12155		The application expects that it will not access the specified range in the near future.
12156		
12157	TYM	The following flags shall be defined for <i>posix_typed_mem_open()</i> :
12158		POSIX_TYPED_MEM_ALLOCATE
12159		Allocate on <i>mmap()</i> .
12160		POSIX_TYPED_MEM_ALLOCATE_CONTIG
12161		Allocate contiguously on <i>mmap()</i> .
12162		POSIX_TYPED_MEM_MAP_ALLOCATABLE
12163		Map on <i>mmap()</i> , without affecting allocatability.
12164		
12165		The mode_t , off_t , and size_t types shall be defined as described in <sys/types.h>.
12166	TYM	The <sys/mman.h> header shall define the structure posix_typed_mem_info , which includes at
12167		least the following member:
12168		size_t posix_tmi_length Maximum length which may be allocated
12169		from a typed memory object.
12170		
12171		The following shall be declared as functions and may also be defined as macros. Function
12172		prototypes shall be provided.
12173	MLR	int mlock(const void *, size_t);
12174	ML	int mlockall(int);
12175	MC3	void *mmap(void *, size_t, int, int, int, off_t);
12176	MPR	int mprotect(void *, size_t, int);
12177	MF SIO	int msync(void *, size_t, int);
12178	MLR	int munlock(const void *, size_t);
12179	ML	int munlockall(void);
12180	MC3	int munmap(void *, size_t);
12181	ADV	int posix_madvise(void *, size_t, int);
12182	TYM	int posix_mem_offset(const void *restrict, size_t, off_t *restrict,
12183		size_t *restrict, int *restrict);
12184		int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);
12185		int posix_typed_mem_open(const char *, int, int);
12186	SHM	int shm_open(const char *, int, mode_t);
12187		int shm_unlink(const char *);
12188		

12189 **APPLICATION USAGE**

12190 None.

12191 **RATIONALE**

12192 None.

12193 **FUTURE DIRECTIONS**

12194 None.

12195 **SEE ALSO**

12196 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *mlock()*, *mlockall()*,
 12197 *mmap()*, *mprotect()*, *msync()*, *munlock()*, *munlockall()*, *munmap()*, *posix_mem_offset()*,
 12198 *posix_typed_mem_get_info()*, *posix_typed_mem_open()*, *shm_open()*, *shm_unlink()*

12199 **CHANGE HISTORY**

12200 First released in Issue 4, Version 2.

12201 **Issue 5**

12202 Updated for alignment with the POSIX Realtime Extension.

12203 **Issue 6**

12204 The <sys/mman.h> header is marked as dependent on support for either the Memory Mapped
 12205 Files, Process Memory Locking, or Shared Memory Objects options.

12206 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 12207 • The TYM margin code is added to the list of margin codes for the <sys/mman.h> header line,
 12208 as well as for other lines.
- 12209 • The POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG, and
 12210 POSIX_TYPED_MEM_MAP_ALLOCATABLE flags are added.
- 12211 • The **posix_tmi_length** structure is added.
- 12212 • The *posix_mem_offset()*, *posix_typed_mem_get_info()*, and *posix_typed_mem_open()* functions
 12213 are added.

12214 The **restrict** keyword is added to the prototype for *posix_mem_offset()*.12215 IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for *posix_madvise()*.

12216 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/16 is applied, correcting margin code and
 12217 shading errors for the *mlock()* and *munlock()* functions. |

12218 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code |
 12219 for the *mmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM). |

12220 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code |
 12221 for the *munmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM). |

12222 **NAME**

12223 sys/msg.h — XSI message queue structures

12224 **SYNOPSIS**

12225 xsi #include <sys/msg.h>

12226

12227 **DESCRIPTION**12228 The <sys/msg.h> header shall define the following data types through **typedef**:12229 **msgqnum_t** Used for the number of messages in the message queue.12230 **msglen_t** Used for the number of bytes allowed in a message queue.12231 These types shall be unsigned integer types that are able to store values at least as large as a type
12232 **unsigned short**.

12233 The <sys/msg.h> header shall define the following constant as a message operation flag:

12234 MSG_NOERROR No error if big message.

12235 The **msqid_ds** structure shall contain the following members:

12236	struct ipc_perm	msg_perm	Operation permission structure.
12237	msgqnum_t	msg_qnum	Number of messages currently on queue.
12238	msglen_t	msg_qbytes	Maximum number of bytes allowed on queue.
12239	pid_t	msg_lspid	Process ID of last <i>msgsnd()</i> .
12240	pid_t	msg_lrpid	Process ID of last <i>msgrcv()</i> .
12241	time_t	msg_stime	Time of last <i>msgsnd()</i> .
12242	time_t	msg_rtime	Time of last <i>msgrcv()</i> .
12243	time_t	msg_ctime	Time of last change.

12244 The **pid_t**, **time_t**, **key_t**, **size_t**, and **ssize_t** types shall be defined as described in <sys/types.h>.12245 The following shall be declared as functions and may also be defined as macros. Function
12246 prototypes shall be provided.

12247	int	msgctl(int, int, struct msqid_ds *);
12248	int	msgget(key_t, int);
12249	ssize_t	msgrcv(int, void *, size_t, long, int);
12250	int	msgsnd(int, const void *, size_t, int);

12251 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12252 **APPLICATION USAGE**

12253 None.

12254 **RATIONALE**

12255 None.

12256 **FUTURE DIRECTIONS**

12257 None.

12258 **SEE ALSO**12259 <sys/ipc.h>, <sys/types.h>, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*12260 **CHANGE HISTORY**

12261 First released in Issue 2. Derived from System V Release 2.0.

12262 **NAME**

12263 sys/resource.h — definitions for XSI resource operations

12264 **SYNOPSIS**

12265 xsi #include <sys/resource.h>

12266

12267 **DESCRIPTION**12268 The <sys/resource.h> header shall define the following symbolic constants as possible values of
12269 the *which* argument of *getpriority()* and *setpriority()*:12270 PRIO_PROCESS Identifies the *who* argument as a process ID.12271 PRIO_PGRP Identifies the *who* argument as a process group ID.12272 PRIO_USER Identifies the *who* argument as a user ID.12273 The following type shall be defined through **typedef**:12274 **rlim_t** Unsigned integer type used for limit values.

12275 The following symbolic constants shall be defined:

12276 RLIM_INFINITY A value of **rlim_t** indicating no limit.12277 RLIM_SAVED_MAX A value of type **rlim_t** indicating an unrepresentable saved hard
12278 limit.12279 RLIM_SAVED_CUR A value of type **rlim_t** indicating an unrepresentable saved soft limit.12280 On implementations where all resource limits are representable in an object of type **rlim_t**,
12281 RLIM_SAVED_MAX and RLIM_SAVED_CUR need not be distinct from RLIM_INFINITY.12282 The following symbolic constants shall be defined as possible values of the *who* parameter of
12283 *getrusage()*:

12284 RUSAGE_SELF Returns information about the current process.

12285 RUSAGE_CHILDREN Returns information about children of the current process.

12286 The <sys/resource.h> header shall define the **rlimit** structure that includes at least the following
12287 members:

12288 rlim_t rlim_cur The current (soft) limit.

12289 rlim_t rlim_max The hard limit.

12290 The <sys/resource.h> header shall define the **rusage** structure that includes at least the following
12291 members:

12292 struct timeval ru_utime User time used.

12293 struct timeval ru_stime System time used.

12294 The **timeval** structure shall be defined as described in <sys/time.h>.12295 The following symbolic constants shall be defined as possible values for the *resource* argument of
12296 *getrlimit()* and *setrlimit()*:12297 RLIMIT_CORE Limit on size of **core** file.

12298 RLIMIT_CPU Limit on CPU time per process.

12299 RLIMIT_DATA Limit on data segment size.

12300 RLIMIT_FSIZE Limit on file size.

12301 RLIMIT_NOFILE Limit on number of open files.

12302 RLIMIT_STACK Limit on stack size.

12303 RLIMIT_AS Limit on address space size.

12304 The following shall be declared as functions and may also be defined as macros. Function
12305 prototypes shall be provided.

12306 int getpriority(int, id_t);

12307 int getrlimit(int, struct rlimit *);

12308 int getrusage(int, struct rusage *);

12309 int setpriority(int, id_t, int);

12310 int setrlimit(int, const struct rlimit *);

12311 The **id_t** type shall be defined through **typedef** as described in <sys/types.h>.

12312 Inclusion of the <sys/resource.h> header may also make visible all symbols from <sys/time.h>.

12313 **APPLICATION USAGE**

12314 None.

12315 **RATIONALE**

12316 None.

12317 **FUTURE DIRECTIONS**

12318 None.

12319 **SEE ALSO**

12320 <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *getpriority()*,
12321 *getrusage()*, *getrlimit()*

12322 **CHANGE HISTORY**

12323 First released in Issue 4, Version 2.

12324 **Issue 5**

12325 Large File System extensions are added.

12326 **NAME**

12327 sys/select.h — select types

12328 **SYNOPSIS**

12329 #include <sys/select.h>

12330 **DESCRIPTION**

12331 The <sys/select.h> header shall define the **timeval** structure that includes at least the following
12332 members:

12333	time_t	tv_sec	Seconds.
12334	suseconds_t	tv_usec	Microseconds.

12335 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.

12336 The **sigset_t** type shall be defined as described in <signal.h>.

12337 The **timespec** structure shall be defined as described in <time.h>.

12338 The <sys/select.h> header shall define the **fd_set** type as a structure.

12339 Each of the following may be declared as a function, or defined as a macro, or both:

12340 **void** *FD_CLR*(int *fd*, **fd_set** **fdset*)

12341 Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.

12342 **int** *FD_ISSET*(int *fd*, **fd_set** **fdset*)

12343 Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by
12344 *fdset*, and 0 otherwise.

12345 **void** *FD_SET*(int *fd*, **fd_set** **fdset*)

12346 Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.

12347 **void** *FD_ZERO*(**fd_set** **fdset*)

12348 Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

12349 If implemented as macros, these may evaluate their arguments more than once, so applications
12350 should ensure that the arguments they supply are never expressions with side effects.

12351 The following shall be defined as a macro:

12352 **FD_SETSIZE**

12353 Maximum number of file descriptors in an **fd_set** structure.

12354 The following shall be declared as functions and may also be defined as macros. Function
12355 prototypes shall be provided.

12356	int	pselect(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12357		const struct timespec *restrict, const sigset_t *restrict);
12358	int	select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12359		struct timeval *restrict);

12360 Inclusion of the <sys/select.h> header may make visible all symbols from the headers
12361 <signal.h>, <sys/time.h>, and <time.h>.

12362 **APPLICATION USAGE**

12363 None.

12364 **RATIONALE**

12365 None.

12366 **FUTURE DIRECTIONS**

12367 None.

12368 **SEE ALSO**

12369 **<signal.h>**, **<sys/time.h>**, **<sys/types.h>**, **<time.h>**, the System Interfaces volume of
12370 IEEE Std 1003.1-2001, *pselect()*, *select()*

12371 **CHANGE HISTORY**

12372 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

12373 The requirement for the **fd_set** structure to have a member *fds_bits* has been removed as per The
12374 Open Group Base Resolution bwg2001-005.

12375 **NAME**

12376 sys/sem.h — XSI semaphore facility

12377 **SYNOPSIS**

12378 xsi #include <sys/sem.h>

12379

12380 **DESCRIPTION**

12381 The <sys/sem.h> header shall define the following constants and structures.

12382 Semaphore operation flags:

12383 SEM_UNDO Set up adjust on exit entry.

12384 Command definitions for the *semctl()* function shall be provided as follows:12385 GETNCNT Get *semncnt*.12386 GETPID Get *sempid*.12387 GETVAL Get *semval*.12388 GETALL Get all cases of *semval*.12389 GETZCNT Get *semzcnt*.12390 SETVAL Set *semval*.12391 SETALL Set all cases of *semval*.12392 The **semid_ds** structure shall contain the following members:

12393 struct ipc_perm sem_perm Operation permission structure.

12394 unsigned short sem_nsems Number of semaphores in set.

12395 time_t sem_otime Last *semop()* time.12396 time_t sem_ctime Last time changed by *semctl()*.12397 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.12398 A semaphore shall be represented by an anonymous structure containing the following
12399 members:

12400 unsigned short semval Semaphore value.

12401 pid_t sempid Process ID of last operation.

12402 unsigned short semncnt Number of processes waiting for *semval*
12403 to become greater than current value.12404 unsigned short semzcnt Number of processes waiting for *semval*
12405 to become 0.12406 The **sembuf** structure shall contain the following members:

12407 unsigned short sem_num Semaphore number.

12408 short sem_op Semaphore operation.

12409 short sem_flg Operation flags.

12410 The following shall be declared as functions and may also be defined as macros. Function
12411 prototypes shall be provided.

12412 int semctl(int, int, int, ...);

12413 int semget(key_t, int, int);

12414 int semop(int, struct sembuf *, size_t);

12415 In addition, all of the symbols from **<sys/ipc.h>** shall be defined when this header is included.

12416 **APPLICATION USAGE**

12417 None.

12418 **RATIONALE**

12419 None.

12420 **FUTURE DIRECTIONS**

12421 None.

12422 **SEE ALSO**

12423 **<sys/ipc.h>**, **<sys/types.h>**, *semctl()*, *semget()*, *semop()*

12424 **CHANGE HISTORY**

12425 First released in Issue 2. Derived from System V Release 2.0.

12426 NAME

12427 sys/shm.h — XSI shared memory facility

12428 SYNOPSIS

12429 xsi #include <sys/shm.h>

12430

12431 DESCRIPTION

12432 The <sys/shm.h> header shall define the following symbolic constants:

12433 SHM_RDONLY Attach read-only (else read-write).

12434 SHM_RND Round attach address to SHMLBA.

12435 The <sys/shm.h> header shall define the following symbolic value:

12436 SHMLBA Segment low boundary address multiple.

12437 The following data types shall be defined through **typedef**:

12438 **shmatt_t** Unsigned integer used for the number of current attaches that must be able to
12439 store values at least as large as a type **unsigned short**.

12440 The **shmid_ds** structure shall contain the following members:

12441	struct ipc_perm	shm_perm	Operation permission structure.
12442	size_t	shm_segsz	Size of segment in bytes.
12443	pid_t	shm_lpid	Process ID of last shared memory operation.
12444	pid_t	shm_cpid	Process ID of creator.
12445	shmatt_t	shm_nattch	Number of current attaches.
12446	time_t	shm_atime	Time of last <i>shmat()</i> .
12447	time_t	shm_dtime	Time of last <i>shmdt()</i> .
12448	time_t	shm_ctime	Time of last change by <i>shmctl()</i> .

12449 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.

12450 The following shall be declared as functions and may also be defined as macros. Function
12451 prototypes shall be provided.

```
12452 void *shmat(int, const void *, int);
12453 int shmctl(int, int, struct shmid_ds *);
12454 int shmdt(const void *);
12455 int shmget(key_t, size_t, int);
```

12456 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12457 APPLICATION USAGE

12458 None.

12459 RATIONALE

12460 None.

12461 FUTURE DIRECTIONS

12462 None.

12463 SEE ALSO

12464 <sys/ipc.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *shmat()*,
12465 *shmctl()*, *shmdt()*, *shmget()*

12466 **CHANGE HISTORY**

12467 First released in Issue 2. Derived from System V Release 2.0.

12468 **Issue 5**

12469 The type of *shm_segsz* is changed from **int** to **size_t**.

12470 **NAME**

12471 sys/socket.h — main sockets header

12472 **SYNOPSIS**

12473 #include <sys/socket.h>

12474 **DESCRIPTION**

12475 The <sys/socket.h> header shall define the type **socklen_t**, which is an integer type of width of
12476 at least 32 bits; see APPLICATION USAGE.

12477 The <sys/socket.h> header shall define the unsigned integer type **sa_family_t**.

12478 The <sys/socket.h> header shall define the **sockaddr** structure that includes at least the
12479 following members:

12480 sa_family_t sa_family Address family.
12481 char sa_data[] Socket address (variable-length data).

12482 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,
12483 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.

12484 The <sys/socket.h> header shall define the **sockaddr_storage** structure. This structure shall be:

- 12485 • Large enough to accommodate all supported protocol-specific address structures
- 12486 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to protocol-
12487 specific address structures and used to access the fields of those structures without
12488 alignment problems

12489 The **sockaddr_storage** structure shall contain at least the following members:

12490 sa_family_t ss_family

12491 When a **sockaddr_storage** structure is cast as a **sockaddr** structure, the *ss_family* field of the
12492 **sockaddr_storage** structure shall map onto the *sa_family* field of the **sockaddr** structure. When a
12493 **sockaddr_storage** structure is cast as a protocol-specific address structure, the *ss_family* field
12494 shall map onto a field of that structure that is of type **sa_family_t** and that identifies the
12495 protocol's address family.

12496 The <sys/socket.h> header shall define the **msghdr** structure that includes at least the following
12497 members:

12498	void	*msg_name	Optional address.
12499	socklen_t	msg_namelen	Size of address.
12500	struct iovec	*msg_iov	Scatter/gather array.
12501	int	msg_iovlen	Members in <i>msg_iov</i> .
12502	void	*msg_control	Ancillary data; see below.
12503	socklen_t	msg_controllen	Ancillary data buffer <i>len</i> .
12504	int	msg_flags	Flags on received message.

12505 The **msghdr** structure is used to minimize the number of directly supplied parameters to the
12506 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the
12507 *recvmsg()* function and *value* only for the *sendmsg()* function.

12508 The **iovec** structure shall be defined as described in <sys/uio.h>.

12509 The <sys/socket.h> header shall define the **cmsg_hdr** structure that includes at least the following
12510 members:

12511	socklen_t	cmsg_len	Data byte count, including the cmsg_hdr .
12512	int	cmsg_level	Originating protocol.

12513 `int` `cmsg_type` Protocol-specific type.

12514 The **cmsghdr** structure is used for storage of ancillary data object information.

12515 Ancillary data consists of a sequence of pairs, each consisting of a **cmsghdr** structure followed
12516 by a data array. The data array contains the ancillary data message, and the **cmsghdr** structure
12517 contains descriptive information that allows an application to correctly parse the data.

12518 The values for `cmsg_level` shall be legal values for the `level` argument to the `getsockopt()` and
12519 `setsockopt()` functions. The system documentation shall specify the `cmsg_type` definitions for the
12520 supported protocols.

12521 Ancillary data is also possible at the socket level. The <sys/socket.h> header defines the
12522 following macro for use as the `cmsg_type` value when `cmsg_level` is `SOL_SOCKET`:

12523 `SCM_RIGHTS` Indicates that the data array contains the access rights to be sent or
12524 received.

12525 The <sys/socket.h> header defines the following macros to gain access to the data arrays in the
12526 ancillary data associated with a message header:

12527 `MSG_DATA(cmsg)`
12528 If the argument is a pointer to a **cmsghdr** structure, this macro shall return an unsigned
12529 character pointer to the data array associated with the **cmsghdr** structure.

12530 `MSG_NXTHDR(mhdr,cmsg)`
12531 If the first argument is a pointer to a **msghdr** structure and the second argument is a pointer
12532 to a **cmsghdr** structure in the ancillary data pointed to by the `msg_control` field of that
12533 **msghdr** structure, this macro shall return a pointer to the next **cmsghdr** structure, or a null
12534 pointer if this structure is the last **cmsghdr** in the ancillary data.

12535 `MSG_FIRSTHDR(mhdr)`
12536 If the argument is a pointer to a **msghdr** structure, this macro shall return a pointer to the
12537 first **cmsghdr** structure in the ancillary data associated with this **msghdr** structure, or a null
12538 pointer if there is no ancillary data associated with the **msghdr** structure.

12539 The <sys/socket.h> header shall define the **linger** structure that includes at least the following
12540 members:

12541 `int` `l_onoff` Indicates whether linger option is enabled.
12542 `int` `l_linger` Linger time, in seconds.

12543 The <sys/socket.h> header shall define the following macros, with distinct integer values:

12544 `SOCK_DGRAM` Datagram socket.

12545 RS `SOCK_RAW` Raw Protocol Interface.

12546 `SOCK_SEQPACKET` Sequenced-packet socket.

12547 `SOCK_STREAM` Byte-stream socket.

12548 The <sys/socket.h> header shall define the following macro for use as the `level` argument of
12549 `setsockopt()` and `getsockopt()`.

12550 `SOL_SOCKET` Options to be accessed at socket level, not protocol level.

12551 The <sys/socket.h> header shall define the following macros, with distinct integer values, for
12552 use as the `option_name` argument in `getsockopt()` or `setsockopt()` calls:

12553 `SO_ACCEPTCONN` Socket is accepting connections.

12554	SO_BROADCAST	Transmission of broadcast messages is supported.
12555	SO_DEBUG	Debugging information is being recorded.
12556	SO_DONTROUTE	Bypass normal routing.
12557	SO_ERROR	Socket error status.
12558	SO_KEEPALIVE	Connections are kept alive with periodic messages.
12559	SO_LINGER	Socket lingers on close.
12560	SO_OOBINLINE	Out-of-band data is transmitted in line.
12561	SO_RCVBUF	Receive buffer size.
12562	SO_RCVLOWAT	Receive “low water mark”.
12563	SO_RCVTIMEO	Receive timeout.
12564	SO_REUSEADDR	Reuse of local addresses is supported.
12565	SO_SNDBUF	Send buffer size.
12566	SO_SNDLOWAT	Send “low water mark”.
12567	SO_SNDTIMEO	Send timeout.
12568	SO_TYPE	Socket type.
12569	The <sys/socket.h> header shall define the following macro as the maximum <i>backlog</i> queue length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
12570		
12571	SOMAXCONN	The maximum <i>backlog</i> queue length.
12572	The <sys/socket.h> header shall define the following macros, with distinct integer values, for use as the valid values for the <i>msg_flags</i> field in the msghdr structure, or the <i>flags</i> parameter in <i>recvfrom()</i> , <i>recvmsg()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
12573		
12574		
12575	MSG_CTRUNC	Control data truncated.
12576	MSG_DONTROUTE	Send without using routing tables.
12577	MSG_EOR	Terminates a record (if supported by the protocol).
12578	MSG_OOB	Out-of-band data.
12579	MSG_PEEK	Leave received data in queue.
12580	MSG_TRUNC	Normal data truncated.
12581	MSG_WAITALL	Attempt to fill the read buffer.
12582	The <sys/socket.h> header shall define the following macros, with distinct integer values:	
12583	AF_INET	Internet domain sockets for use with IPv4 addresses.
12584	IP6 AF_INET6	Internet domain sockets for use with IPv6 addresses.
12585	AF_UNIX	UNIX domain sockets.
12586	AF_UNSPEC	Unspecified.
12587	The <sys/socket.h> header shall define the following macros, with distinct integer values:	
12588	SHUT_RD	Disables further receive operations.

```

12589      SHUT_RDWR          Disables further send and receive operations.
12590      SHUT_WR           Disables further send operations.
12591      The following shall be declared as functions and may also be defined as macros. Function
12592      prototypes shall be provided.
12593      int      accept(int, struct sockaddr *restrict, socklen_t *restrict);
12594      int      bind(int, const struct sockaddr *, socklen_t);
12595      int      connect(int, const struct sockaddr *, socklen_t);
12596      int      getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
12597      int      getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
12598      int      getsockopt(int, int, int, void *restrict, socklen_t *restrict);
12599      int      listen(int, int);
12600      ssize_t  recv(int, void *, size_t, int);
12601      ssize_t  recvfrom(int, void *restrict, size_t, int,
12602      struct sockaddr *restrict, socklen_t *restrict);
12603      ssize_t  recvmsg(int, struct msghdr *, int);
12604      ssize_t  send(int, const void *, size_t, int);
12605      ssize_t  sendmsg(int, const struct msghdr *, int);
12606      ssize_t  sendto(int, const void *, size_t, int, const struct sockaddr *,
12607      socklen_t);
12608      int      setsockopt(int, int, int, const void *, socklen_t);
12609      int      shutdown(int, int);
12610      int      socket(int, int, int);
12611      int      socketatmark(int);
12612      int      socketpair(int, int, int, int[2]);

```

12613 Inclusion of <sys/socket.h> may also make visible all symbols from <sys/uio.h>.

12614 APPLICATION USAGE

12615 To forestall portability problems, it is recommended that applications not use values larger than
12616 $2^{31} - 1$ for the **socklen_t** type.

12617 The **sockaddr_storage** structure solves the problem of declaring storage for automatic variables
12618 which is both large enough and aligned enough for storing the socket address data structure of
12619 any family. For example, code with a file descriptor and without the context of the address
12620 family can pass a pointer to a variable of this type, where a pointer to a socket address structure
12621 is expected in calls such as *getpeername()*, and determine the address family by accessing the
12622 received content after the call.

12623 The example below illustrates a data structure which aligns on a 64-bit boundary. An
12624 implementation-defined field *_ss_align* following *_ss_pad1* is used to force a 64-bit alignment
12625 which covers proper alignment good enough for needs of at least **sockaddr_in6** (IPv6) and
12626 **sockaddr_in** (IPv4) address data structures. The size of padding field *_ss_pad1* depends on the
12627 chosen alignment boundary. The size of padding field *_ss_pad2* depends on the value of overall
12628 size chosen for the total size of the structure. This size and alignment are represented in the
12629 above example by implementation-defined (not required) constants *_SS_MAXSIZE* (chosen
12630 value 128) and *_SS_ALIGNMENT* (with chosen value 8). Constants *_SS_PAD1SIZE* (derived
12631 value 6) and *_SS_PAD2SIZE* (derived value 112) are also for illustration and not required. The
12632 implementation-defined definitions and structure field names above start with an underscore to
12633 denote implementation private name space. Portable code is not expected to access or reference
12634 those fields or constants.

```

12635      /*
12636      *   Desired design of maximum size and alignment.

```

```

12637     */
12638     #define _SS_MAXSIZE 128
12639     /* Implementation-defined maximum size. */
12640     #define _SS_ALIGNSIZE (sizeof(int64_t))
12641     /* Implementation-defined desired alignment. */
12642     /*
12643     * Definitions used for sockaddr_storage structure paddings design.
12644     */
12645     #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
12646     #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t) + \
12647         _SS_PAD1SIZE + _SS_ALIGNSIZE))
12648     struct sockaddr_storage {
12649         sa_family_t ss_family; /* Address family. */
12650     /*
12651     * Following fields are implementation-defined.
12652     */
12653         char _ss_pad1[_SS_PAD1SIZE];
12654         /* 6-byte pad; this is to make implementation-defined
12655         pad up to alignment field that follows explicit in
12656         the data structure. */
12657         int64_t _ss_align; /* Field to force desired structure
12658         storage alignment. */
12659         char _ss_pad2[_SS_PAD2SIZE];
12660         /* 112-byte pad to achieve desired size,
12661         _SS_MAXSIZE value minus size of ss_family
12662         __ss_pad1, __ss_align fields is 112. */
12663     };

```

12664 RATIONALE

12665 None.

12666 FUTURE DIRECTIONS

12667 None.

12668 SEE ALSO

12669 <sys/uio.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *accept()*, *bind()*, *connect()*,
12670 *getpeername()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*,
12671 *sendmsg()*, *sendto()*, *setsockopt()*, *shutdown()*, *socket()*, *socketpair()*

12672 CHANGE HISTORY

12673 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

12674 The **restrict** keyword is added to the prototypes for *accept()*, *getpeername()*, *getsockname()*,
12675 *getsockopt()*, and *recvfrom()*.

12676 **NAME**

12677 sys/stat.h — data returned by the stat() function

12678 **SYNOPSIS**

12679 #include <sys/stat.h>

12680 **DESCRIPTION**12681 The <sys/stat.h> header shall define the structure of the data returned by the functions *fstat()*,
12682 *lstat()*, and *stat()*.12683 The **stat** structure shall contain at least the following members:

12684	dev_t	st_dev	Device ID of device containing file.
12685	ino_t	st_ino	File serial number.
12686	mode_t	st_mode	Mode of file (see below).
12687	nlink_t	st_nlink	Number of hard links to the file.
12688	uid_t	st_uid	User ID of file.
12689	gid_t	st_gid	Group ID of file.
12690 XSI	dev_t	st_rdev	Device ID (if file is character or block special).
12691	off_t	st_size	For regular files, the file size in bytes.
12692			For symbolic links, the length in bytes of the
12693			pathname contained in the symbolic link.
12694 SHM			For a shared memory object, the length in bytes.
12695 TYM			For a typed memory object, the length in bytes.
12696			For other file types, the use of this field is
12697			unspecified.
12698	time_t	st_atime	Time of last access.
12699	time_t	st_mtime	Time of last data modification.
12700	time_t	st_ctime	Time of last status change.
12701 XSI	blksize_t	st_blksize	A file system-specific preferred I/O block size for
12702			this object. In some file system types, this may
12703			vary from file to file.
12704	blkcnt_t	st_blocks	Number of blocks allocated for this object.
12705			

12706 The *st_ino* and *st_dev* fields taken together uniquely identify the file within the system. The
 12707 **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types shall be
 12708 defined as described in <sys/types.h>. Times shall be given in seconds since the Epoch.

12709 Unless otherwise specified, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*,
 12710 *st_ctime*, and *st_mtime* shall have meaningful values for all file types defined in
 12711 IEEE Std 1003.1-2001.

12712 For symbolic links, the *st_mode* member shall contain meaningful information, which can be
 12713 used with the file type macros described below, that take a *mode* argument. The *st_size* member
 12714 shall contain the length, in bytes, of the pathname contained in the symbolic link. File mode bits
 12715 and the contents of the remaining members of the **stat** structure are unspecified. The value
 12716 returned in the *st_size* field shall be the length of the contents of the symbolic link, and shall not
 12717 count a trailing null if one is present.

12718 The following symbolic names for the values of type **mode_t** shall also be defined.

12719 File type:

12720 XSI	S_IFMT	Type of file.
12721	S_IFBLK	Block special.

12722		S_IFCHR	Character special.
12723		S_IFIFO	FIFO special.
12724		S_IFREG	Regular.
12725		S_IFDIR	Directory.
12726		S_IFLNK	Symbolic link.
12727		S_IFSOCK	Socket.
12728	File mode bits:		
12729	S_IRWXU	Read, write, execute/search by owner.	
12730		S_IRUSR	Read permission, owner.
12731		S_IWUSR	Write permission, owner.
12732		S_IXUSR	Execute/search permission, owner.
12733	S_IRWXG	Read, write, execute/search by group.	
12734		S_IRGRP	Read permission, group.
12735		S_IWGRP	Write permission, group.
12736		S_IXGRP	Execute/search permission, group.
12737	S_IRWXO	Read, write, execute/search by others.	
12738		S_IROTH	Read permission, others.
12739		S_IWOTH	Write permission, others.
12740		S_IXOTH	Execute/search permission, others.
12741	S_ISUID	Set-user-ID on execution.	
12742	S_ISGID	Set-group-ID on execution.	
12743	XSI	S_ISVTX	On directories, restricted deletion flag.
12744	The bits defined by S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH,		
12745	XSI	S_IWOTH, S_IXOTH, S_ISUID, S_ISGID, and S_ISVTX shall be unique.	
12746	S_IRWXU is the bitwise-inclusive OR of S_IRUSR, S_IWUSR, and S_IXUSR.		
12747	S_IRWXG is the bitwise-inclusive OR of S_IRGRP, S_IWGRP, and S_IXGRP.		
12748	S_IRWXO is the bitwise-inclusive OR of S_IROTH, S_IWOTH, and S_IXOTH.		
12749	Implementations may OR other implementation-defined bits into S_IRWXU, S_IRWXG, and		
12750	S_IRWXO, but they shall not overlap any of the other bits defined in this volume of		
12751	IEEE Std 1003.1-2001. The <i>file permission bits</i> are defined to be those corresponding to the		
12752	bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO.		
12753	The following macros shall be provided to test whether a file is of the specified type. The value		
12754	<i>m</i> supplied to the macros is the value of <i>st_mode</i> from a stat structure. The macro shall evaluate		
12755	to a non-zero value if the test is true; 0 if the test is false.		
12756	S_ISBLK(<i>m</i>)	Test for a block special file.	
12757	S_ISCHR(<i>m</i>)	Test for a character special file.	

12758 **S_ISDIR(*m*)** Test for a directory.

12759 **S_ISFIFO(*m*)** Test for a pipe or FIFO special file.

12760 **S_ISREG(*m*)** Test for a regular file.

12761 **S_ISLNK(*m*)** Test for a symbolic link.

12762 **S_ISSOCK(*m*)** Test for a socket.

12763 The implementation may implement message queues, semaphores, or shared memory objects as distinct file types. The following macros shall be provided to test whether a file is of the specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

12769 **S_TYPEISMQ(*buf*)** Test for a message queue.

12770 **S_TYPEISSEM(*buf*)** Test for a semaphore.

12771 **S_TYPEISSHM(*buf*)** Test for a shared memory object.

12772 TYP The implementation may implement typed memory objects as distinct file types, and the following macro shall test whether a file is of the specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

12777 **S_TYPEISTMO(*buf*)** Test macro for a typed memory object.

12779 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

12781 `int       chmod(const char *, mode_t);`

12782 `int       fchmod(int, mode_t);`

12783 `int       fstat(int, struct stat *);`

12784 `int       lstat(const char *restrict, struct stat *restrict);`

12785 `int       mkdir(const char *, mode_t);`

12786 `int       mkfifo(const char *, mode_t);`

12787 XSI `int       mknod(const char *, mode_t, dev_t);`

12788 `int       stat(const char *restrict, struct stat *restrict);`

12789 `mode_t   umask(mode_t);`

12790 APPLICATION USAGE

12791 Use of the macros is recommended for determining the type of a file.

12792 RATIONALE

12793 A conforming C-language application must include <sys/stat.h> for functions that have arguments or return values of type **mode_t**, so that symbolic values for that type can be used. An alternative would be to require that these constants are also defined by including <sys/types.h>.

12797 The **S_ISUID** and **S_ISGID** bits may be cleared on any write, not just on *open()*, as some historical implementations do.

12799 System calls that update the time entry fields in the **stat** structure must be documented by the implementors. POSIX-conforming systems should not update the time entry fields for functions listed in the System Interfaces volume of IEEE Std 1003.1-2001 unless the standard requires that

12802 they do, except in the case of documented extensions to the standard.

12803 Note that *st_dev* must be unique within a Local Area Network (LAN) in a “system” made up of
12804 multiple computers’ file systems connected by a LAN.

12805 Networked implementations of a POSIX-conforming system must guarantee that all files visible
12806 within the file tree (including parts of the tree that may be remotely mounted from other
12807 machines on the network) on each individual processor are uniquely identified by the
12808 combination of the *st_ino* and *st_dev* fields.

12809 The unit for the *st_blocks* member of the **stat** structure is not defined within IEEE Std 1003.1-2001.
12810 In some implementations it is 512 bytes. It may differ on a file system basis. There is no
12811 correlation between values of the *st_blocks* and *st_blksize*, and the *f_bsize* (from <sys/statvfs.h>
12812 structure members.

12813 Traditionally, some implementations defined the multiplier for *st_blocks* in <sys/param.h> as the
12814 symbol DEV_BSIZE.

12815 **FUTURE DIRECTIONS**

12816 No new S_IFMT symbolic names for the file type values of **mode_t** will be defined by
12817 IEEE Std 1003.1-2001; if new file types are required, they will only be testable through *S_ISxx()*
12818 or *S_TYPEISxxx()* macros instead.

12819 **SEE ALSO**

12820 <sys/statvfs.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *chmod()*,
12821 *fchmod()*, *fstat()*, *lstat()*, *mkdir()*, *mkfifo()*, *mknod()*, *stat()*, *umask()*

12822 **CHANGE HISTORY**

12823 First released in Issue 1. Derived from Issue 1 of the SVID.

12824 **Issue 5**

12825 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

12826 The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from
12827 **long** to **blkcnt_t**.

12828 **Issue 6**

12829 The *S_TYPEISMQ()*, *S_TYPEISSEM()*, and *S_TYPEISSHM()* macros are unconditionally
12830 mandated.

12831 The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize_t**
12832 and **blkcnt_t** have been described.

12833 The following new requirements on POSIX implementations derive from alignment with the
12834 Single UNIX Specification:

12835 • The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.

12836 *S_IFSOCK* and *S_ISSOCK* are added for sockets.

12837 The description of **stat** structure members is changed to reflect contents when file type is a
12838 symbolic link.

12839 The test macro *S_TYPEISTMO* is added for alignment with IEEE Std 1003.1j-2000.

12840 The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*.

12841 The *lstat()* function is made mandatory.

12842 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/17 is applied, adding text regarding the
12843 *st_blocks* member of the **stat** structure to the RATIONALE.

12844 **NAME**

12845 sys/statvfs.h — VFS File System information structure

12846 **SYNOPSIS**

12847 xSI #include <sys/statvfs.h>

12848

12849 **DESCRIPTION**12850 The <sys/statvfs.h> header shall define the **statvfs** structure that includes at least the following
12851 members:

12852	unsigned long	f_bsize	File system block size.
12853	unsigned long	f_frsize	Fundamental file system block size.
12854	fsblkcnt_t	f_blocks	Total number of blocks on file system in units of <i>f_frsize</i> .
12855	fsblkcnt_t	f_bfree	Total number of free blocks.
12856	fsblkcnt_t	f_bavail	Number of free blocks available to non-privileged process.
12857			
12858	fsfilcnt_t	f_files	Total number of file serial numbers.
12859	fsfilcnt_t	f_ffree	Total number of free file serial numbers.
12860	fsfilcnt_t	f_favail	Number of file serial numbers available to non-privileged process.
12861			
12862	unsigned long	f_fsid	File system ID.
12863	unsigned long	f_flag	Bit mask of <i>f_flag</i> values.
12864	unsigned long	f_namemax	Maximum filename length.

12865 The **fsblkcnt_t** and **fsfilcnt_t** types shall be defined as described in <sys/types.h>.12866 The following flags for the *f_flag* member shall be defined:

12867 ST_RDONLY Read-only file system.

12868 ST_NOSUID Does not support the semantics of the ST_ISUID and ST_ISGID file mode bits. |

12869 The following shall be declared as functions and may also be defined as macros. Function
12870 prototypes shall be provided.

12871 int statvfs(const char *restrict, struct statvfs *restrict);

12872 int fstatvfs(int, struct statvfs *);

12873 **APPLICATION USAGE**

12874 None.

12875 **RATIONALE**

12876 None.

12877 **FUTURE DIRECTIONS**

12878 None.

12879 **SEE ALSO**12880 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *fstatvfs()*, *statvfs()*12881 **CHANGE HISTORY**

12882 First released in Issue 4, Version 2.

12883 **Issue 5**12884 The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type
12885 of *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

12886 **Issue 6**

12887 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt_t**
12888 and **fsfilcnt_t** have been described.

12889 The **restrict** keyword is added to the prototype for *statvfs()*.

12890 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/18 is applied, changing the description of |
12891 ST_NOSUID from “Does not support *setuid()/setgid()* semantics” to “Does not support the |
12892 semantics of the ST_ISUID and ST_ISGID file mode bits”. |

12893 **NAME**

12894 sys/time.h — time types

12895 **SYNOPSIS**

12896 xSI #include <sys/time.h>

12897

12898 **DESCRIPTION**12899 The <sys/time.h> header shall define the **timeval** structure that includes at least the following
12900 members:

12901 time_t tv_sec Seconds.

12902 suseconds_t tv_usec Microseconds.

12903 The <sys/time.h> header shall define the **itimerval** structure that includes at least the following
12904 members:

12905 struct timeval it_interval Timer interval.

12906 struct timeval it_value Current value.

12907 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.12908 The **fd_set** type shall be defined as described in <sys/select.h>.12909 The <sys/time.h> header shall define the following values for the *which* argument of *getitimer()*
12910 and *setitimer()*:

12911 ITIMER_REAL Decrements in real time.

12912 ITIMER_VIRTUAL Decrements in process virtual time.

12913 ITIMER_PROF Decrements both in process virtual time and when the system is running
12914 on behalf of the process.

12915 The following shall be defined as described in <sys/select.h>:

12916 FD_CLR()

12917 FD_ISSET()

12918 FD_SET()

12919 FD_ZERO()

12920 FD_SETSIZE

12921 The following shall be declared as functions and may also be defined as macros. Function
12922 prototypes shall be provided.

12923 int getitimer(int, struct itimerval *);

12924 int gettimeofday(struct timeval *restrict, void *restrict);

12925 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12926 struct timeval *restrict);12927 int setitimer(int, const struct itimerval *restrict,
12928 struct itimerval *restrict);12929 int utimes(const char *, const struct timeval [2]); (**LEGACY**)12930 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>
12931 header.

12932 **APPLICATION USAGE**

12933 None.

12934 **RATIONALE**

12935 None.

12936 **FUTURE DIRECTIONS**

12937 None.

12938 **SEE ALSO**

12939 <sys/select.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *getitimer()*,
12940 *gettimeofday()*, *select()*, *setitimer()*

12941 **CHANGE HISTORY**

12942 First released in Issue 4, Version 2.

12943 **Issue 5**

12944 The type of *tv_usec* is changed from **long** to **suseconds_t**.

12945 **Issue 6**

12946 The **restrict** keyword is added to the prototypes for *gettimeofday()*, *select()*, and *setitimer()*.

12947 The note is added that inclusion of this header may also make symbols visible from
12948 <sys/select.h>.

12949 The *utimes()* function is marked LEGACY.

12950 **NAME**

12951 sys/timeb.h — additional definitions for date and time

12952 **SYNOPSIS**

12953 xSI #include <sys/timeb.h>

12954

12955 **DESCRIPTION**12956 The <sys/timeb.h> header shall define the **timeb** structure that includes at least the following
12957 members:

12958	time_t	time	The seconds portion of the current time.
12959	unsigned short	millitm	The milliseconds portion of the current time.
12960	short	timezone	The local timezone in minutes west of Greenwich.
12961	short	dstflag	TRUE if Daylight Savings Time is in effect.

12962 The **time_t** type shall be defined as described in <sys/types.h>.12963 The following shall be declared as a function and may also be defined as a macro. A function
12964 prototype shall be provided.12965 int ftime(struct timeb *); (**LEGACY**)12966 **APPLICATION USAGE**

12967 None.

12968 **RATIONALE**

12969 None.

12970 **FUTURE DIRECTIONS**

12971 None.

12972 **SEE ALSO**

12973 <sys/types.h>, <time.h>

12974 **CHANGE HISTORY**

12975 First released in Issue 4, Version 2.

12976 **Issue 6**12977 The *ftime()* function is marked LEGACY.

12978 **NAME**

12979 sys/times.h — file access and modification times structure

12980 **SYNOPSIS**

12981 #include <sys/times.h>

12982 **DESCRIPTION**

12983 The <sys/times.h> header shall define the structure **tms**, which is returned by *times()* and
12984 includes at least the following members:

12985 clock_t tms_utime User CPU time.
12986 clock_t tms_stime System CPU time.
12987 clock_t tms_cutime User CPU time of terminated child processes.
12988 clock_t tms_cstime System CPU time of terminated child processes.

12989 The **clock_t** type shall be defined as described in <sys/types.h>.

12990 The following shall be declared as a function and may also be defined as a macro. A function
12991 prototype shall be provided.

12992 clock_t times(struct tms *);

12993 **APPLICATION USAGE**

12994 None.

12995 **RATIONALE**

12996 None.

12997 **FUTURE DIRECTIONS**

12998 None.

12999 **SEE ALSO**

13000 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *times()*

13001 **CHANGE HISTORY**

13002 First released in Issue 1. Derived from Issue 1 of the SVID.

13003 **NAME**

13004 sys/types.h — data types

13005 **SYNOPSIS**

13006 #include <sys/types.h>

13007 **DESCRIPTION**

13008 The <sys/types.h> header shall include definitions for at least the following types:

13009	blkcnt_t	Used for file block counts.
13010	blksize_t	Used for block sizes.
13011 XSI	clock_t	Used for system times in clock ticks or CLOCKS_PER_SEC; see
13012		<time.h>.
13013 TMR	clockid_t	Used for clock ID type in the clock and timer functions.
13014	dev_t	Used for device IDs.
13015 XSI	fsblkcnt_t	Used for file system block counts.
13016 XSI	fsfilcnt_t	Used for file system file counts.
13017	gid_t	Used for group IDs.
13018 XSI	id_t	Used as a general identifier; can be used to contain at least a pid_t ,
13019		uid_t , or gid_t .
13020	ino_t	Used for file serial numbers.
13021 XSI	key_t	Used for XSI interprocess communication.
13022	mode_t	Used for some file attributes.
13023	nlink_t	Used for link counts.
13024	off_t	Used for file sizes.
13025	pid_t	Used for process IDs and process group IDs.
13026 THR	pthread_attr_t	Used to identify a thread attribute object.
13027 BAR	pthread_barrier_t	Used to identify a barrier.
13028 BAR	pthread_barrierattr_t	Used to define a barrier attributes object.
13029 THR	pthread_cond_t	Used for condition variables.
13030 THR	pthread_condattr_t	Used to identify a condition attribute object.
13031 THR	pthread_key_t	Used for thread-specific data keys.
13032 THR	pthread_mutex_t	Used for mutexes.
13033 THR	pthread_mutexattr_t	Used to identify a mutex attribute object.
13034 THR	pthread_once_t	Used for dynamic package initialization.
13035 THR	pthread_rwlock_t	Used for read-write locks.
13036 THR	pthread_rwlockattr_t	Used for read-write lock attributes.
13037 SPI	pthread_spinlock_t	Used to identify a spin lock.
13038 THR	pthread_t	Used to identify a thread.

13039		size_t	Used for sizes of objects.
13040		ssize_t	Used for a count of bytes or an error indication.
13041	XSI	suseconds_t	Used for time in microseconds.
13042		time_t	Used for time in seconds.
13043	TMR	timer_t	Used for timer ID returned by <i>timer_create()</i> .
13044	TRC	trace_attr_t	Used to identify a trace stream attributes object.
13045	TRC	trace_event_id_t	Used to identify a trace event type.
13046	TRC TEF	trace_event_set_t	Used to identify a trace event type set.
13047	TRC	trace_id_t	Used to identify a trace stream.
13048		uid_t	Used for user IDs.
13049	XSI	useconds_t	Used for time in microseconds.
13050	All of the types shall be defined as arithmetic types of an appropriate length, with the following		
13051	exceptions:		
13052	XSI	key_t	
13053	THR	pthread_attr_t	
13054	BAR	pthread_barrier_t	
13055		pthread_barrierattr_t	
13056	THR	pthread_cond_t	
13057		pthread_condattr_t	
13058		pthread_key_t	
13059		pthread_mutex_t	
13060		pthread_mutexattr_t	
13061		pthread_once_t	
13062		pthread_rwlock_t	
13063		pthread_rwlockattr_t	
13064	SPI	pthread_spinlock_t	
13065	TRC	trace_attr_t	
13066		trace_event_id_t	
13067	TRC TEF	trace_event_set_t	
13068	TRC	trace_id_t	
13069			
13070	Additionally:		
13071	• mode_t shall be an integer type.		
13072	• nlink_t , uid_t , gid_t , and id_t shall be integer types.		
13073	• blkcnt_t and off_t shall be signed integer types.		
13074	XSI	• fsblkcnt_t , fsfilcnt_t , and ino_t shall be defined as unsigned integer types.	
13075	• size_t shall be an unsigned integer type.		
13076	• blksize_t , pid_t , and ssize_t shall be signed integer types.		
13077	• time_t and clock_t shall be integer or real-floating types.		
13078	XSI	The type ssize_t shall be capable of storing values at least in the range $[-1, \{\text{SSIZE\_MAX}\}]$. The	
13079	type useconds_t shall be an unsigned integer type capable of storing values at least in the range		
13080	$[0, 1\,000\,000]$. The type suseconds_t shall be a signed integer type capable of storing values at		

13081 least in the range `[-1, 1 000 000]`.

13082 The implementation shall support one or more programming environments in which the widths
13083 of **blksize_t**, **pid_t**, **size_t**, **ssize_t**, **suseconds_t**, and **useconds_t** are no greater than the width of
13084 type **long**. The names of these programming environments can be obtained using the *confstr()*
13085 function or the *getconf* utility.

13086 There are no defined comparison or assignment operators for the following types:

13087	THR	pthread_attr_t
13088	BAR	pthread_barrier_t
13089		pthread_barrierattr_t
13090	THR	pthread_cond_t
13091		pthread_condattr_t
13092		pthread_mutex_t
13093		pthread_mutexattr_t
13094		pthread_rwlock_t
13095		pthread_rwlockattr_t
13096	SPI	pthread_spinlock_t
13097	TRC	trace_attr_t
13098		

13099 APPLICATION USAGE

13100 None.

13101 RATIONALE

13102 None.

13103 FUTURE DIRECTIONS

13104 None.

13105 SEE ALSO

13106 <time.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *confstr()*, the Shell and Utilities
13107 volume of IEEE Std 1003.1-2001, *getconf*

13108 CHANGE HISTORY

13109 First released in Issue 1. Derived from Issue 1 of the SVID.

13110 Issue 5

13111 The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.

13112 The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.

13113 Large File System extensions are added.

13114 Updated for alignment with the POSIX Threads Extension.

13115 Issue 6

13116 The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for
13117 alignment with IEEE Std 1003.1j-2000.

13118 The margin code is changed from XSI to THR for the **pthread_rwlock_t** and
13119 **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads
13120 option. The threads types are marked THR.

13121 NAME

13122 sys/uio.h — definitions for vector I/O operations

13123 SYNOPSIS

13124 xSI #include <sys/uio.h>

13125

13126 DESCRIPTION

13127 The <sys/uio.h> header shall define the **iovec** structure that includes at least the following
13128 members:

13129 void *iov_base Base address of a memory region for input or output.
13130 size_t iov_len The size of the memory pointed to by *iov_base*.

13131 The <sys/uio.h> header uses the **iovec** structure for scatter/gather I/O.

13132 The **ssize_t** and **size_t** types shall be defined as described in <sys/types.h>.

13133 The following shall be declared as functions and may also be defined as macros. Function
13134 prototypes shall be provided.

13135 ssize_t readv(int, const struct iovec *, int);
13136 ssize_t writev(int, const struct iovec *, int);

13137 APPLICATION USAGE

13138 The implementation can put a limit on the number of scatter/gather elements which can be
13139 processed in one call. The symbol {IOV_MAX} defined in <limits.h> should always be used to
13140 learn about the limits instead of assuming a fixed value.

13141 RATIONALE

13142 Traditionally, the maximum number of scatter/gather elements the system can process in one
13143 call were described by the symbolic value {UIO_MAXIOV}. In IEEE Std 1003.1-2001 this value is
13144 replaced by the constant {IOV_MAX} which can be found in <limits.h>.

13145 FUTURE DIRECTIONS

13146 None.

13147 SEE ALSO

13148 <limits.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *read()*, *write()*

13149 CHANGE HISTORY

13150 First released in Issue 4, Version 2.

13151 Issue 6

13152 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13153 **NAME**

13154 sys/un.h — definitions for UNIX domain sockets

13155 **SYNOPSIS**

13156 #include <sys/un.h>

13157 **DESCRIPTION**13158 The <sys/un.h> header shall define the **sockaddr_un** structure that includes at least the
13159 following members:13160 sa_family_t sun_family Address family.
13161 char sun_path[] Socket pathname.13162 The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Values of this
13163 type shall be cast by applications to **struct sockaddr** for use with socket functions.13164 The **sa_family_t** type shall be defined as described in <sys/socket.h>.13165 **APPLICATION USAGE**13166 The size of *sun_path* has intentionally been left undefined. This is because different
13167 implementations use different sizes. For example, 4.3 BSD uses a size of 108, and 4.4 BSD uses a
13168 size of 104. Since most implementations originate from BSD versions, the size is typically in the
13169 range 92 to 108.13170 Applications should not assume a particular length for *sun_path* or assume that it can hold
13171 {_POSIX_PATH_MAX} characters (255).13172 **RATIONALE**

13173 None.

13174 **FUTURE DIRECTIONS**

13175 None.

13176 **SEE ALSO**13177 <sys/socket.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *bind()*, *socket()*,
13178 *socketpair()*13179 **CHANGE HISTORY**

13180 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13181 **NAME**

13182 sys/utsname.h — system name structure

13183 **SYNOPSIS**

13184 #include <sys/utsname.h>

13185 **DESCRIPTION**

13186 The <sys/utsname.h> header shall define the structure **utsname** which shall include at least the
13187 following members:

13188	char	sysname[]	Name of this implementation of the operating system.
13189	char	nodename[]	Name of this node within an implementation-defined communications network.
13190			
13191	char	release[]	Current release level of this implementation.
13192	char	version[]	Current version level of this release.
13193	char	machine[]	Name of the hardware type on which the system is running.

13194 The character arrays are of unspecified size, but the data stored in them shall be terminated by a
13195 null byte.

13196 The following shall be declared as a function and may also be defined as a macro:

13197 int uname(struct utsname *);

13198 **APPLICATION USAGE**

13199 None.

13200 **RATIONALE**

13201 None.

13202 **FUTURE DIRECTIONS**

13203 None.

13204 **SEE ALSO**

13205 The System Interfaces volume of IEEE Std 1003.1-2001, *uname()*

13206 **CHANGE HISTORY**

13207 First released in Issue 1. Derived from Issue 1 of the SVID.

13208 **NAME**

13209 sys/wait.h — declarations for waiting

13210 **SYNOPSIS**

13211 #include <sys/wait.h>

13212 **DESCRIPTION**13213 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

13214 WNOHANG Do not hang if no status is available; return immediately.

13215 WUNTRACED Report status of stopped child process.

13216 The <sys/wait.h> header shall define the following macros for analysis of process status values:

13217 WEXITSTATUS Return exit status.

13218 XSI WIFCONTINUED True if child has been continued.

13219 WIFEXITED True if child exited normally.

13220 WIFSIGNALED True if child exited due to uncaught signal.

13221 WIFSTOPPED True if child is currently stopped.

13222 WSTOPSIG Return signal number that caused process to stop.

13223 WTERMSIG Return signal number that caused process to terminate.

13224 XSI The following symbolic constants shall be defined as possible values for the *options* argument to
13225 *waitid()*:

13226 WEXITED Wait for processes that have exited.

13227 WSTOPPED Status is returned for any child that has stopped upon receipt of a signal.

13228 WCONTINUED Status is returned for any child that was stopped and has been continued.

13229 WNOHANG Return immediately if there are no children to wait for.

13230 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state.13231 The type **idtype_t** shall be defined as an enumeration type whose possible values shall include
13232 at least the following:

13233 P_ALL

13234 P_PID

13235 P_PGID

13236

13237 The **id_t** and **pid_t** types shall be defined as described in <sys/types.h>.13238 XSI The **siginfo_t** type shall be defined as described in <signal.h>.13239 The **rusage** structure shall be defined as described in <sys/resource.h>.13240 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h> and
13241 <sys/resource.h>.13242 The following shall be declared as functions and may also be defined as macros. Function
13243 prototypes shall be provided.

13244 pid_t wait(int *);

13245 XSI int waitid(idtype_t, id_t, siginfo_t *, int);

13246 pid_t waitpid(pid_t, int *, int);

13247 **APPLICATION USAGE**

13248 None.

13249 **RATIONALE**

13250 None.

13251 **FUTURE DIRECTIONS**

13252 None.

13253 **SEE ALSO**

13254 <signal.h>, <sys/resource.h>, <sys/types.h>, the System Interfaces volume of
13255 IEEE Std 1003.1-2001, *wait()*, *waitid()*

13256 **CHANGE HISTORY**

13257 First released in Issue 3.

13258 Included for alignment with the POSIX.1-1988 standard.

13259 **Issue 6**

13260 The *wait3()* function is removed.

13261 **NAME**

13262 syslog.h — definitions for system error logging

13263 **SYNOPSIS**

13264 xSI #include <syslog.h>

13265

13266 **DESCRIPTION**13267 The <syslog.h> header shall define the following symbolic constants, zero or more of which may
13268 be OR'ed together to form the *logopt* option of *openlog()*:

13269 LOG_PID Log the process ID with each message.

13270 LOG_CONS Log to the system console on error.

13271 LOG_NDELAY Connect to syslog daemon immediately.

13272 LOG_ODELAY Delay open until *syslog()* is called.

13273 LOG_NOWAIT Do not wait for child processes.

13274 The following symbolic constants shall be defined as possible values of the *facility* argument to
13275 *openlog()*:

13276 LOG_KERN Reserved for message generated by the system.

13277 LOG_USER Message generated by a process.

13278 LOG_MAIL Reserved for message generated by mail system.

13279 LOG_NEWS Reserved for message generated by news system.

13280 LOG_UUCP Reserved for message generated by UUCP system.

13281 LOG_DAEMON Reserved for message generated by system daemon.

13282 LOG_AUTH Reserved for message generated by authorization daemon.

13283 LOG_CRON Reserved for message generated by clock daemon.

13284 LOG_LPR Reserved for message generated by printer system.

13285 LOG_LOCAL0 Reserved for local use.

13286 LOG_LOCAL1 Reserved for local use.

13287 LOG_LOCAL2 Reserved for local use.

13288 LOG_LOCAL3 Reserved for local use.

13289 LOG_LOCAL4 Reserved for local use.

13290 LOG_LOCAL5 Reserved for local use.

13291 LOG_LOCAL6 Reserved for local use.

13292 LOG_LOCAL7 Reserved for local use.

13293 The following shall be declared as macros for constructing the *maskpri* argument to *setlogmask()*.
13294 The following macros expand to an expression of type **int** when the argument *pri* is an
13295 expression of type **int**:13296 LOG_MASK(*pri*) A mask for priority *pri*.13297 The following constants shall be defined as possible values for the *priority* argument of *syslog()*:

13298	LOG_EMERG	A panic condition was reported to all processes.
13299	LOG_ALERT	A condition that should be corrected immediately.
13300	LOG_CRIT	A critical condition.
13301	LOG_ERR	An error message.
13302	LOG_WARNING	A warning message.
13303	LOG_NOTICE	A condition requiring special handling.
13304	LOG_INFO	A general information message.
13305	LOG_DEBUG	A message useful for debugging programs.
13306	The following shall be declared as functions and may also be defined as macros. Function	
13307	prototypes shall be provided.	
13308	void	closelog(void);
13309	void	openlog(const char *, int, int);
13310	int	setlogmask(int);
13311	void	syslog(int, const char *, ...);
13312	APPLICATION USAGE	
13313	None.	
13314	RATIONALE	
13315	None.	
13316	FUTURE DIRECTIONS	
13317	None.	
13318	SEE ALSO	
13319	The System Interfaces volume of IEEE Std 1003.1-2001, <i>closelog()</i>	
13320	CHANGE HISTORY	
13321	First released in Issue 4, Version 2.	
13322	Issue 5	
13323	Moved from X/Open UNIX to BASE.	

13324 **NAME**

13325 tar.h — extended tar definitions

13326 **SYNOPSIS**

13327 #include <tar.h>

13328 **DESCRIPTION**

13329 The <tar.h> header shall define header block definitions as follows.

13330 General definitions:

13331

13332

13333

13334

13335

13336

Name	Description	Value
TMAGIC	"ustar"	ustar plus null byte.
TMAGLEN	6	Length of the above.
TVERSION	"00"	00 without a null byte.
TVERSLEN	2	Length of the above.

13337 *Typeflag* field definitions:

13338

13339

13340

13341

13342

13343

13344

13345

13346

13347

13348

Name	Description	Value
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Symbolic link.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory.
FIFOTYPE	'6'	FIFO special.
CONTTYPE	'7'	Reserved.

13349 *Mode* field bit definitions (octal):

13350

13351

13352

13353

13354 xsi

13355

13356

13357

13358

13359

13360

13361

13362

13363

Name	Description	Value
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

13364 **APPLICATION USAGE**

13365 None.

13366 **RATIONALE**

13367 None.

13368 **FUTURE DIRECTIONS**

13369 None.

13370 **SEE ALSO**

13371 The Shell and Utilities volume of IEEE Std 1003.1-2001, *pax*

13372 **CHANGE HISTORY**

13373 First released in Issue 3. Derived from the POSIX.1-1988 standard.

13374 **Issue 6**

13375 The SEE ALSO section now refers to *pax* since the Shell and Utilities volume of
13376 IEEE Std 1003.1-2001 no longer contains the *tar* utility.

13377 **NAME**

13378 termios.h — define values for termios

13379 **SYNOPSIS**

13380 #include <termios.h>

13381 **DESCRIPTION**

13382 The <termios.h> header contains the definitions used by the terminal I/O interfaces (see
 13383 Chapter 11 (on page 187) for the structures and names defined).

13384 **The termios Structure**13385 The following data types shall be defined through **typedef**:13386 **cc_t** Used for terminal special characters.13387 **speed_t** Used for terminal baud rates.13388 **tcflag_t** Used for terminal modes.

13389 The above types shall be all unsigned integer types.

13390 The implementation shall support one or more programming environments in which the widths
 13391 of **cc_t**, **speed_t**, and **tcflag_t** are no greater than the width of type **long**. The names of these
 13392 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

13393 The **termios** structure shall be defined, and shall include at least the following members:

13394 tcflag_t c_iflag Input modes.
 13395 tcflag_t c_oflag Output modes.
 13396 tcflag_t c_cflag Control modes.
 13397 tcflag_t c_lflag Local modes.
 13398 cc_t c_cc [NCCS] Control characters.

13399 A definition shall be provided for:

13400 NCCS Size of the array *c_cc* for control characters.13401 The following subscript names for the array *c_cc* shall be defined:

13402

13403

13404

13405

13406

13407

13408

13409

13410

13411

13412

13413

13414

13415

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF	VINTR	EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR		INTR character.
VKILL	VMIN	KILL character.
		MIN value.
VQUIT		QUIT character.
VSTART		START character.
VSTOP	VSUSP	STOP character.
VSUSP		SUSP character.
	VTIME	TIME value.

13416 The subscript values shall be unique, except that the VMIN and VTIME subscripts may have the
 13417 same values as the VEOF and VEOL subscripts, respectively.

13418 The following flags shall be provided.

13419	Input Modes	
13420	The <i>c_iflag</i> field describes the basic terminal input control:	
13421	BRKINT	Signal interrupt on break.
13422	ICRNL	Map CR to NL on input.
13423	IGNBRK	Ignore break condition.
13424	IGNCR	Ignore CR.
13425	IGNPAR	Ignore characters with parity errors.
13426	INLCR	Map NL to CR on input.
13427	INPCK	Enable input parity check.
13428	ISTRIP	Strip character.
13429 XSI	IXANY	Enable any character to restart output.
13430	IXOFF	Enable start/stop input control.
13431	IXON	Enable start/stop output control.
13432	PARMRK	Mark parity errors.

13433	Output Modes	
13434	The <i>c_oflag</i> field specifies the system treatment of output:	
13435	OPOST	Post-process output.
13436 XSI	ONLCR	Map NL to CR-NL on output.
13437	OCRNL	Map CR to NL on output.
13438	ONOCR	No CR output at column 0.
13439	ONLRET	NL performs CR function.
13440	OFILL	Use fill characters for delay.
13441	NLDLY	Select newline delays:
13442	NL0	Newline type 0.
13443	NL1	Newline type 1.
13444	CRDLY	Select carriage-return delays:
13445	CR0	Carriage-return delay type 0.
13446	CR1	Carriage-return delay type 1.
13447	CR2	Carriage-return delay type 2.
13448	CR3	Carriage-return delay type 3.
13449	TABDLY	Select horizontal-tab delays:
13450	TAB0	Horizontal-tab delay type 0.
13451	TAB1	Horizontal-tab delay type 1.
13452	TAB2	Horizontal-tab delay type 2.

13453		TAB3	Expand tabs to spaces.
13454	BSDLY		Select backspace delays:
13455		BS0	Backspace-delay type 0.
13456		BS1	Backspace-delay type 1.
13457	VTDLY		Select vertical-tab delays:
13458		VT0	Vertical-tab delay type 0.
13459		VT1	Vertical-tab delay type 1.
13460	FFDLY		Select form-feed delays:
13461		FF0	Form-feed delay type 0.
13462		FF1	Form-feed delay type 1.

13463 **Baud Rate Selection**

13464 The input and output baud rates are stored in the **termios** structure. These are the valid values
 13465 for objects of type **speed_t**. The following values shall be defined, but not all baud rates need be
 13466 supported by the underlying hardware.

13467	B0	Hang up
13468	B50	50 baud
13469	B75	75 baud
13470	B110	110 baud
13471	B134	134.5 baud
13472	B150	150 baud
13473	B200	200 baud
13474	B300	300 baud
13475	B600	600 baud
13476	B1200	1 200 baud
13477	B1800	1 800 baud
13478	B2400	2 400 baud
13479	B4800	4 800 baud
13480	B9600	9 600 baud
13481	B19200	19 200 baud
13482	B38400	38 400 baud

13483 **Control Modes**

13484 The *c_cflag* field describes the hardware control of the terminal; not all values specified are
 13485 required to be supported by the underlying hardware:

13486 CSIZE Character size:

13487 CS5 5 bits

13488 CS6 6 bits

13489 CS7 7 bits

13490 CS8 8 bits

13491 CSTOPB Send two stop bits, else one.

13492 CREAD Enable receiver.

13493 PARENB Parity enable.

13494 PARODD Odd parity, else even.

13495 HUPCL Hang up on last close.

13496 CLOCAL Ignore modem status lines.

13497 The implementation shall support the functionality associated with the symbols CS7, CS8,
 13498 CSTOPB, PARODD, and PARENB.

13499 **Local Modes**

13500 The *c_lflag* field of the argument structure is used to control various terminal functions:

13501 ECHO Enable echo.

13502 ECHOE Echo erase character as error-correcting backspace.

13503 ECHOK Echo KILL.

13504 ECHONL Echo NL.

13505 ICANON Canonical input (erase and kill processing).

13506 IEXTEN Enable extended input character processing.

13507 ISIG Enable signals.

13508 NOFLSH Disable flush after interrupt or quit.

13509 TOSTOP Send SIGTTOU for background output.

13510 **Attribute Selection**

13511 The following symbolic constants for use with *tcsetattr()* are defined:

13512 TCSANOW Change attributes immediately.

13513 TCSADRAIN Change attributes when output has drained.

13514 TCSAFLUSH Change attributes when output has drained; also flush pending input.

13515 **Line Control**

13516 The following symbolic constants for use with *tcflush()* shall be defined:

13517 TCIFLUSH Flush pending input.

13518 TCIOFLUSH Flush both pending input and untransmitted output.

13519 TCOFLUSH Flush untransmitted output.

13520 The following symbolic constants for use with *tcflow()* shall be defined:

13521 TCIOFF Transmit a STOP character, intended to suspend input data.

13522 TCION Transmit a START character, intended to restart input data.

13523 TCOOFF Suspend output.

13524 TCOON Restart output.

13525 The following shall be declared as functions and may also be defined as macros. Function
13526 prototypes shall be provided.

```
13527 speed_t cfgetispeed(const struct termios *);
13528 speed_t cfgetospeed(const struct termios *);
13529 int      cfsetispeed(struct termios *, speed_t);
13530 int      cfsetospeed(struct termios *, speed_t);
13531 int      tcdrain(int);
13532 int      tcflow(int, int);
13533 int      tcflush(int, int);
13534 int      tcgetattr(int, struct termios *);
13535 xsi pid_t tcgetsid(int);
13536 int      tcsendbreak(int, int);
13537 int      tcsetattr(int, int, const struct termios *);
```

13538 **APPLICATION USAGE**

13539 The following names are reserved for XSI-conformant systems to use as an extension to the
13540 above; therefore strictly conforming applications shall not use them:

13541	CBAUD	EXTB	VDSUSP
13542	DEFECHO	FLUSHO	VLNEXT
13543	ECHOCTL	LOBLK	VREPRINT
13544	ECHOKE	PENDIN	VSTATUS
13545	ECHOPRT	SWTCH	VWERASE
13546	EXTA	VDISCARD	

13547 **RATIONALE**

13548 None.

13549 **FUTURE DIRECTIONS**

13550 None.

13551 **SEE ALSO**

13552 The System Interfaces volume of IEEE Std 1003.1-2001, *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*,
13553 *cfsetospeed()*, *confstr()*, *tcdrain()*, *tcflow()*, *tcflush()*, *tcgetattr()*, *tcgetsid()*, *tcsendbreak()*, *tcsetattr()*,
13554 the Shell and Utilities volume of IEEE Std 1003.1-2001, *getconf*, Chapter 11 (on page 187)

13555 **CHANGE HISTORY**

13556 First released in Issue 3.

13557 Included for alignment with the ISO POSIX-1 standard.

13558 **Issue 6**

13559 The LEGACY symbols IUCLC, OLCUC, and XCASE are removed.

13560 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are
13561 reaffirmed.

13562 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/19 is applied, changing ECHOK to |
13563 ECHOK in the APPLICATION USAGE section. |

13564 **NAME**

13565 tmath.h — type-generic macros

13566 **SYNOPSIS**

13567 #include <tmath.h>

13568 **DESCRIPTION**

13569 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 13570 conflict between the requirements described here and the ISO C standard is unintentional. This
 13571 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13572 The <tmath.h> header shall include the headers <math.h> and <complex.h> and shall define
 13573 several type-generic macros.

13574 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**) or
 13575 *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is
 13576 **double**. For each such function, except *modf()*, there shall be a corresponding type-generic
 13577 macro. The parameters whose corresponding real type is **double** in the function synopsis are
 13578 generic parameters. Use of the macro invokes a function whose corresponding real type and
 13579 type domain are determined by the arguments for the generic parameters.

13580 Use of the macro invokes a function whose generic parameters have the corresponding real type
 13581 determined as follows:

- 13582 • First, if any argument for generic parameters has type **long double**, the type determined is
 13583 **long double**.
- 13584 • Otherwise, if any argument for generic parameters has type **double** or is of integer type, the
 13585 type determined is **double**.
- 13586 • Otherwise, the type determined is **float**.

13587 For each unsuffixed function in the <math.h> header for which there is a function in the
 13588 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic
 13589 macro (for both functions) has the same name as the function in the <math.h> header. The
 13590 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

<math.h> Function	<complex.h> Function	Type-Generic Macro
<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
<i>log()</i>	<i>clog()</i>	<i>log()</i>
<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

13610 If at least one argument for a generic parameter is complex, then use of the macro invokes a
13611 complex function; otherwise, use of the macro invokes a real function.

13612 For each unsuffixed function in the <math.h> header without a *c*-prefixed counterpart in the
13613 <complex.h> header, the corresponding type-generic macro has the same name as the function.
13614 These type-generic macros are:

13615	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
13616	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
13617	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
13618	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
13619	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>
13620	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>
13621	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
13622	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
13623	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
13624	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

13625 If all arguments for generic parameters are real, then use of the macro invokes a real function;
13626 otherwise, use of the macro results in undefined behavior.

13627 For each unsuffixed function in the <complex.h> header that is not a *c*-prefixed counterpart to a
13628 function in the <math.h> header, the corresponding type-generic macro has the same name as
13629 the function. These type-generic macros are:

13630 *carg()*
13631 *cimag()*
13632 *conj()*
13633 *cproj()*
13634 *creal()*

13635 Use of the macro with any real or complex argument invokes a complex function.

13636 APPLICATION USAGE

13637 With the declarations:

```
13638 #include <tgmath.h>
13639 int n;
13640 float f;
13641 double d;
13642 long double ld;
13643 float complex fc;
13644 double complex dc;
13645 long double complex ldc;
```

13646 functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
<i>exp(n)</i>	<i>exp(n)</i> , the function
<i>acosh(f)</i>	<i>acoshf(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>

13653

13654

13655

13656

13657

13658

13659

13660

13661

13662

13663

13664

13665

13666

13667

13668

13669

13670

13671

Macro	Use Invokes
<i>log</i> (<i>fc</i>)	<i>clogf</i> (<i>fc</i>)
<i>sqrt</i> (<i>dc</i>)	<i>csqrt</i> (<i>dc</i>)
<i>pow</i> (<i>ldc</i> , <i>f</i>)	<i>cpowl</i> (<i>ldc</i> , <i>f</i>)
<i>remainder</i> (<i>n</i> , <i>n</i>)	<i>remainder</i> (<i>n</i> , <i>n</i>), the function
<i>nextafter</i> (<i>d</i> , <i>f</i>)	<i>nextafter</i> (<i>d</i> , <i>f</i>), the function
<i>nexttoward</i> (<i>f</i> , <i>ld</i>)	<i>nexttowardf</i> (<i>f</i> , <i>ld</i>)
<i>copysign</i> (<i>n</i> , <i>ld</i>)	<i>copysignl</i> (<i>n</i> , <i>ld</i>)
<i>ceil</i> (<i>fc</i>)	Undefined behavior
<i>rint</i> (<i>dc</i>)	Undefined behavior
<i>fmax</i> (<i>ldc</i> , <i>ld</i>)	Undefined behavior
<i>carg</i> (<i>n</i>)	<i>carg</i> (<i>n</i>), the function
<i>cproj</i> (<i>f</i>)	<i>cprojf</i> (<i>f</i>)
<i>creal</i> (<i>d</i>)	<i>creal</i> (<i>d</i>), the function
<i>cimag</i> (<i>ld</i>)	<i>cimagl</i> (<i>ld</i>)
<i>cabs</i> (<i>fc</i>)	<i>cabsf</i> (<i>fc</i>)
<i>carg</i> (<i>dc</i>)	<i>carg</i> (<i>dc</i>), the function
<i>cproj</i> (<i>ldc</i>)	<i>cprojl</i> (<i>ldc</i>)

13672 **RATIONALE**

13673

13674

13675

13676

13677

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '*'. For example, with a type-generic `cos()` macro, the expression `cos((float)x)` will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

13678

13679

13680

13681

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to `nexttoward()`, whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

13682

13683

13684

The term “type-generic” was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran’s generic functions than to C++ overloading.

13685

13686

The macros are placed in their own header in order not to silently break old programs that include the **<math.h>** header; for example, with:

13687

```
printf ("%e", sin(x))
```

13688

13689

`modf(double, double *)` is excluded because no way was seen to make it safe without complicating the type resolution.

13690

13691

13692

The implementation might, as an extension, endow appropriate ones of the macros that IEEE Std 1003.1-2001 specifies only for real arguments with the ability to invoke the complex functions.

13693

13694

13695

IEEE Std 1003.1-2001 does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for `sqrt()`, for example, could be implemented with:

13696

13697

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

13698

13699

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

13700	The great majority of existing C programs are expected to be unaffected when the <tgmath.h>
13701	header is included instead of the <math.h> or <complex.h> headers. Generic macros are similar
13702	to the ISO /IEC 9899:1999 standard library masking macros, though the semantic types of return
13703	values differ.
13704	The ability to overload on integer as well as floating types would have been useful for some
13705	functions; for example, <i>copysign()</i> . Overloading with different numbers of arguments would
13706	have allowed reusing names; for example, <i>remainder()</i> for <i>remquo()</i> . However, these facilities
13707	would have complicated the specification; and their natural consistent use, such as for a floating
13708	<i>abs()</i> or a two-argument <i>atan()</i> , would have introduced further inconsistencies with the
13709	ISO /IEC 9899:1999 standard for insufficient benefit.
13710	The ISO C standard in no way limits the implementation's options for efficiency, including
13711	inlining library functions.
13712	FUTURE DIRECTIONS
13713	None.
13714	SEE ALSO
13715	<math.h>, <complex.h>, the System Interfaces volume of IEEE Std 1003.1-2001, <i>cabs()</i> , <i>fabs()</i> ,
13716	<i>modf()</i>
13717	CHANGE HISTORY
13718	First released in Issue 6. Included for alignment with the ISO /IEC 9899:1999 standard.

13719 **NAME**

13720 time.h — time types

13721 **SYNOPSIS**

13722 #include <time.h>

13723 **DESCRIPTION**

13724 CX Some of the functionality described on this reference page extends the ISO C standard.
 13725 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 13726 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 13727 symbols in this header.

13728 The <time.h> header shall declare the structure **tm**, which shall include at least the following
 13729 members:

13730	int	tm_sec	Seconds [0,60].
13731	int	tm_min	Minutes [0,59].
13732	int	tm_hour	Hour [0,23].
13733	int	tm_mday	Day of month [1,31].
13734	int	tm_mon	Month of year [0,11].
13735	int	tm_year	Years since 1900.
13736	int	tm_wday	Day of week [0,6] (Sunday =0).
13737	int	tm_yday	Day of year [0,365].
13738	int	tm_isdst	Daylight Savings flag.

13739 The value of *tm_isdst* shall be positive if Daylight Savings Time is in effect, 0 if Daylight Savings
 13740 Time is not in effect, and negative if the information is not available.

13741 The <time.h> header shall define the following symbolic names:

13742	NULL	Null pointer constant.
13743	CLOCKS_PER_SEC	A number used to convert the value returned by the <i>clock()</i> function into
13744		seconds.

13745	TMR CPT	CLOCK_PROCESS_CPUTIME_ID
13746		The identifier of the CPU-time clock associated with the process making a
13747		<i>clock()</i> or <i>timer*()</i> function call.

13748	TMR TCT	CLOCK_THREAD_CPUTIME_ID
13749		The identifier of the CPU-time clock associated with the thread making a
13750		<i>clock()</i> or <i>timer*()</i> function call.

13751 TMR The <time.h> header shall declare the structure **timespec**, which has at least the following
 13752 members:

13753	time_t	tv_sec	Seconds.
13754	long	tv_nsec	Nanoseconds.

13755 The <time.h> header shall also declare the **itimerspec** structure, which has at least the following
 13756 members:

13757	struct timespec	it_interval	Timer period.
13758	struct timespec	it_value	Timer expiration.

13759 The following manifest constants shall be defined:

13760	CLOCK_REALTIME	The identifier of the system-wide realtime clock.

13761	TIMER_ABSTIME	Flag indicating time is absolute. For functions taking timer objects, this
13762		refers to the clock associated with the timer.

13763 MON	CLOCK_MONOTONIC
13764	The identifier for the system-wide monotonic clock, which is defined as a
13765	clock whose value cannot be set via <i>clock_settime()</i> and which cannot
13766	have backward clock jumps. The maximum possible clock jump shall be
13767	implementation-defined.
13768 TMR	The clock_t , size_t , time_t , clockid_t , and timer_t types shall be defined as described in
13769	<sys/types.h>.
13770 XSI	Although the value of CLOCKS_PER_SEC is required to be 1 million on all XSI-conformant
13771	systems, it may be variable on other systems, and it should not be assumed that
13772	CLOCKS_PER_SEC is a compile-time constant.
13773 XSI	The <time.h> header shall provide a declaration for <i>getdate_err</i> .
13774	The following shall be declared as functions and may also be defined as macros. Function
13775	prototypes shall be provided.
13776	char *asctime(const struct tm *);
13777 TSF	char *asctime_r(const struct tm *restrict, char *restrict);
13778	clock_t clock(void);
13779 CPT	int clock_getcpuclockid(pid_t, clockid_t *);
13780 TMR	int clock_getres(clockid_t, struct timespec *);
13781	int clock_gettime(clockid_t, struct timespec *);
13782 CS	int clock_nanosleep(clockid_t, int, const struct timespec *,
13783	struct timespec *);
13784 TMR	int clock_settime(clockid_t, const struct timespec *);
13785	char *ctime(const time_t *);
13786 TSF	char *ctime_r(const time_t *, char *);
13787	double difftime(time_t, time_t);
13788 XSI	struct tm *getdate(const char *);
13789	struct tm *gmtime(const time_t *);
13790 TSF	struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
13791	struct tm *localtime(const time_t *);
13792 TSF	struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
13793	time_t mktime(struct tm *);
13794 TMR	int nanosleep(const struct timespec *, struct timespec *);
13795	size_t strftime(char *restrict, size_t, const char *restrict,
13796	const struct tm *restrict);
13797 XSI	char *strptime(const char *restrict, const char *restrict,
13798	struct tm *restrict);
13799	time_t time(time_t *);
13800 TMR	int timer_create(clockid_t, struct sigevent *restrict,
13801	timer_t *restrict);
13802	int timer_delete(timer_t);
13803	int timer_gettime(timer_t, struct itimerspec *);
13804	int timer_getoverrun(timer_t);
13805	int timer_settime(timer_t, int, const struct itimerspec *restrict,
13806	struct itimerspec *restrict);
13807 CX	void tzset(void);
13808	

13809 The following shall be declared as variables:

```
13810 XSI    extern int    daylight;
13811        extern long   timezone;
13812 CX      extern char  *tzname[];
13813
```

13814 CX Inclusion of the <time.h> header may make visible all symbols from the <signal.h> header.

13815 APPLICATION USAGE

13816 The range [0,60] for *tm_sec* allows for the occasional leap second.

13817 *tm_year* is a signed value; therefore, years before 1900 may be represented.

13818 To obtain the number of clock ticks per second returned by the *times()* function, applications
13819 should call *sysconf(_SC_CLK_TCK)*.

13820 RATIONALE

13821 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of
13822 UTC does not permit double leap seconds, so all mention of double leap seconds has been
13823 removed, and the range shortened from the former [0,61] seconds seen in previous versions of
13824 POSIX.

13825 FUTURE DIRECTIONS

13826 None.

13827 SEE ALSO

13828 <signal.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *asctime()*,
13829 *clock()*, *clock_getcpuclockid()*, *clock_getres()*, *clock_nanosleep()*, *ctime()*, *difftime()*, *getdate()*,
13830 *gmtime()*, *localtime()*, *mktime()*, *nanosleep()*, *strftime()*, *strptime()*, *sysconf()*, *time()*, *timer_create()*,
13831 *timer_delete()*, *timer_getoverrun()*, *tzname*, *tzset()*, *utime()*

13832 CHANGE HISTORY

13833 First released in Issue 1. Derived from Issue 1 of the SVID.

13834 Issue 5

13835 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
13836 Threads Extension.

13837 Issue 6

13838 The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid_t**
13839 and **timer_t** have been described.

13840 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 13841 • The POSIX timer-related functions are marked as part of the Timers option.

13842 The symbolic name CLK_TCK is removed. Application usage is added describing how its
13843 equivalent functionality can be obtained using *sysconf()*.

13844 The *clock_getcpuclockid()* function and manifest constants CLOCK_PROCESS_CPUTIME_ID and
13845 CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std 1003.1d-1999.

13846 The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep()* function are added for
13847 alignment with IEEE Std 1003.1j-2000.

13848 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- 13849 • The range for seconds is changed from [0,61] to [0,60].
- 13850 • The **restrict** keyword is added to the prototypes for *asctime_r()*, *gmtime_r()*, *localtime_r()*,
13851 *strftime()*, *strptime()*, *timer_create()*, and *timer_settime()*.

13852 IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the
13853 <signal.h> header may be made visible when the <time.h> header is included.
13854 Extensions beyond the ISO C standard are marked.

13855 **NAME**

13856 trace.h — tracing

13857 **SYNOPSIS**

13858 TRC #include <trace.h>

13859

13860 **DESCRIPTION**

13861 The <trace.h> header shall define the **posix_trace_event_info** structure that includes at least the
 13862 following members:

13863	trace_event_id_t	posix_event_id
13864	pid_t	posix_pid
13865	void	*posix_prog_address
13866	int	posix_truncation_status
13867	struct timespec	posix_timestamp
13868 THR	pthread_t	posix_thread_id

13869

13870 The <trace.h> header shall define the **posix_trace_status_info** structure that includes at least the
 13871 following members:

13872	int	posix_stream_status
13873	int	posix_stream_full_status
13874	int	posix_stream_overrun_status
13875 TRL	int	posix_stream_flush_status
13876	int	posix_stream_flush_error
13877	int	posix_log_overrun_status
13878	int	posix_log_full_status

13879

13880 The <trace.h> header shall define the following symbols:

13881	POSIX_TRACE_ALL_EVENTS
13882 TRL	POSIX_TRACE_APPEND
13883 TRI	POSIX_TRACE_CLOSE_FOR_CHILD
13884 TEF	POSIX_TRACE_FILTER
13885 TRL	POSIX_TRACE_FLUSH
13886	POSIX_TRACE_FLUSH_START
13887	POSIX_TRACE_FLUSH_STOP
13888	POSIX_TRACE_FLUSHING
13889	POSIX_TRACE_FULL
13890	POSIX_TRACE_LOOP
13891	POSIX_TRACE_NO_OVERRUN
13892 TRL	POSIX_TRACE_NOT_FLUSHING
13893	POSIX_TRACE_NOT_FULL
13894 TRI	POSIX_TRACE_INHERITED
13895	POSIX_TRACE_NOT_TRUNCATED
13896	POSIX_TRACE_OVERFLOW
13897	POSIX_TRACE_OVERRUN
13898	POSIX_TRACE_RESUME
13899	POSIX_TRACE_RUNNING
13900	POSIX_TRACE_START
13901	POSIX_TRACE_STOP
13902	POSIX_TRACE_SUSPENDED
13903	POSIX_TRACE_SYSTEM_EVENTS

13904 POSIX_TRACE_TRUNCATED_READ
 13905 POSIX_TRACE_TRUNCATED_RECORD
 13906 POSIX_TRACE_UNNAMED_USER_EVENT
 13907 POSIX_TRACE_UNTIL_FULL
 13908 POSIX_TRACE_WOPID_EVENTS

13909 The following types shall be defined as described in <sys/types.h>:

13910 **trace_attr_t**
 13911 **trace_id_t**
 13912 **trace_event_id_t**
 13913 TEF **trace_event_set_t**
 13914

13915 The following shall be declared as functions and may also be defined as macros. Function
 13916 prototypes shall be provided.

```
13917 int  posix_trace_attr_destroy(trace_attr_t *);
13918 int  posix_trace_attr_getclockres(const trace_attr_t *,
13919                                  struct timespec *);
13920 int  posix_trace_attr_getcreatetime(const trace_attr_t *,
13921                                    struct timespec *);
13922 int  posix_trace_attr_getgenversion(const trace_attr_t *, char *);
13923 TRI int  posix_trace_attr_getinherited(const trace_attr_t *restrict,
13924                                       int *restrict);
13925 TRL int  posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict,
13926                                           int *restrict);
13927 int  posix_trace_attr_getlogsize(const trace_attr_t *restrict,
13928                                  size_t *restrict);
13929 int  posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict,
13930                                     size_t *restrict);
13931 int  posix_trace_attr_getmaxsystemeventsz(const trace_attr_t *restrict,
13932                                           size_t *restrict);
13933 int  posix_trace_attr_getmaxusereventsiz(const trace_attr_t *restrict,
13934                                           size_t, size_t *restrict);
13935 int  posix_trace_attr_getname(const trace_attr_t *, char *);
13936 int  posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict,
13937                                           int *restrict);
13938 int  posix_trace_attr_getstreamsize(const trace_attr_t *restrict,
13939                                     size_t *restrict);
13940 int  posix_trace_attr_init(trace_attr_t *);
13941 TRI int  posix_trace_attr_setinherited(trace_attr_t *, int);
13942 TRL int  posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
13943 int  posix_trace_attr_setlogsize(trace_attr_t *, size_t);
13944 int  posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
13945 int  posix_trace_attr_setname(trace_attr_t *, const char *);
13946 int  posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
13947 int  posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
13948 int  posix_trace_clear(trace_id_t);
13949 TRL int  posix_trace_close(trace_id_t);
13950 int  posix_trace_create(pid_t, const trace_attr_t *restrict,
13951                       trace_id_t *restrict);
13952 TRL int  posix_trace_create_withlog(pid_t, const trace_attr_t *restrict,
13953                                   int, trace_id_t *restrict);
```

```

13954 void posix_trace_event(trace_event_id_t, const void *restrict, size_t);
13955 int  posix_trace_eventid_equal(trace_id_t, trace_event_id_t,
13956                               trace_event_id_t);
13957 int  posix_trace_eventid_get_name(trace_id_t, trace_event_id_t, char *);
13958 int  posix_trace_eventid_open(const char *restrict,
13959                               trace_event_id_t *restrict);
13960 TEF int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
13961 int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
13962 int  posix_trace_eventset_empty(trace_event_set_t *);
13963 int  posix_trace_eventset_fill(trace_event_set_t *, int);
13964 int  posix_trace_eventset_ismember(trace_event_id_t,
13965                                   const trace_event_set_t *restrict, int *restrict);
13966 int  posix_trace_eventtypelist_getnext_id(trace_id_t,
13967                                           trace_event_id_t *restrict, int *restrict);
13968 int  posix_trace_eventtypelist_rewind(trace_id_t);
13969 TRL int  posix_trace_flush(trace_id_t);
13970 int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
13971 TEF int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
13972 int  posix_trace_get_status(trace_id_t,
13973                             struct posix_trace_status_info *);
13974 int  posix_trace_getnext_event(trace_id_t,
13975                                struct posix_trace_event_info *restrict, void *restrict,
13976                                size_t, size_t *restrict, int *restrict);
13977 TRL int  posix_trace_open(int, trace_id_t *);
13978 int  posix_trace_rewind(trace_id_t);
13979 TEF int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
13980 int  posix_trace_shutdown(trace_id_t);
13981 int  posix_trace_start(trace_id_t);
13982 int  posix_trace_stop(trace_id_t);
13983 TMO int  posix_trace_timedgetnext_event(trace_id_t,
13984                                         struct posix_trace_event_info *restrict, void *restrict,
13985                                         size_t, size_t *restrict, int *restrict,
13986                                         const struct timespec *restrict);
13987 TEF int  posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
13988                                         trace_event_id_t *restrict);
13989 int  posix_trace_trygetnext_event(trace_id_t,
13990                                   struct posix_trace_event_info *restrict, void *restrict, size_t,
13991                                   size_t *restrict, int *restrict);

```

13992 APPLICATION USAGE

13993 None.

13994 RATIONALE

13995 None.

13996 FUTURE DIRECTIONS

13997 None.

13998 SEE ALSO

13999 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.11, Tracing,
14000 *posix_trace_attr_destroy()*, *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*,
14001 *posix_trace_attr_getgenversion()*, *posix_trace_attr_getinherited()*, *posix_trace_attr_getlogfullpolicy()*,
14002 *posix_trace_attr_getlogsize()*, *posix_trace_attr_getmaxdatasize()*,
14003 *posix_trace_attr_getmaxsystemeventsizesize()*, *posix_trace_attr_getmaxusereventsizesize()*,

```

14004     posix_trace_attr_getname(), posix_trace_attr_getstreamfullpolicy(), posix_trace_attr_getstreamsize(),
14005     posix_trace_attr_init(), posix_trace_attr_setinherited(), posix_trace_attr_setlogfullpolicy(),
14006     posix_trace_attr_setlogsize(), posix_trace_attr_setmaxdatasize(), posix_trace_attr_setname(),
14007     posix_trace_attr_setstreamsize(), posix_trace_attr_setstreamfullpolicy(), posix_trace_clear(),
14008     posix_trace_close(), posix_trace_create(), posix_trace_create_withlog(), posix_trace_event(),
14009     posix_trace_eventid_equal(), posix_trace_eventid_get_name(), posix_trace_eventid_open(),
14010     posix_trace_eventtypelist_getnext_id(), posix_trace_eventtypelist_rewind(),
14011     posix_trace_eventset_add(), posix_trace_eventset_del(), posix_trace_eventset_empty(),
14012     posix_trace_eventset_fill(), posix_trace_eventset_ismember(), posix_trace_flush(),
14013     posix_trace_get_attr(), posix_trace_get_filter(), posix_trace_get_status(), posix_trace_getnext_event(),
14014     posix_trace_open(), posix_trace_rewind(), posix_trace_set_filter(), posix_trace_shutdown(),
14015     posix_trace_start(), posix_trace_stop(), posix_trace_timedgetnext_event(),
14016     posix_trace_trid_eventid_open(), posix_trace_trygetnext_event()

```

14017 CHANGE HISTORY

```

14018     First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

14019     IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/40 is applied, adding the TRL margin |
14020     code to the posix_trace_flush() function, for alignment with the System Interfaces volume of |
14021     IEEE Std 1003.1-2001. |

```

14022 **NAME**

14023 ucontext.h — user context

14024 **SYNOPSIS**

14025 xSI #include <ucontext.h>

14026

14027 **DESCRIPTION**14028 The <ucontext.h> header shall define the **mcontext_t** type through **typedef**.14029 The <ucontext.h> header shall define the **ucontext_t** type as a structure that shall include at least
14030 the following members:

14031	ucontext_t	*uc_link	Pointer to the context that is resumed
14032			when this context returns.
14033	sigset_t	uc_sigmask	The set of signals that are blocked when this
14034			context is active.
14035	stack_t	uc_stack	The stack used by this context.
14036	mcontext_t	uc_mcontext	A machine-specific representation of the saved
14037			context.

14038 The types **sigset_t** and **stack_t** shall be defined as in <signal.h>.14039 The following shall be declared as functions and may also be defined as macros, Function
14040 prototypes shall be provided.

```

14041       int  getcontext(ucontext_t *);
14042       int  setcontext(const ucontext_t *);
14043       void makecontext(ucontext_t *, void (*)(void), int, ...);
14044       int  swapcontext(ucontext_t *restrict, const ucontext_t *restrict);

```

14045 **APPLICATION USAGE**

14046 None.

14047 **RATIONALE**

14048 None.

14049 **FUTURE DIRECTIONS**

14050 None.

14051 **SEE ALSO**14052 <signal.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *getcontext()*, *makecontext()*,
14053 *sigaction()*, *sigprocmask()*, *sigaltstack()*14054 **CHANGE HISTORY**

14055 First released in Issue 4, Version 2.

14056 **NAME**

14057 ulimit.h — ulimit commands

14058 **SYNOPSIS**

14059 xSI #include <ulimit.h>

14060

14061 **DESCRIPTION**

14062 The <ulimit.h> header shall define the symbolic constants used by the *ulimit()* function.

14063 Symbolic constants:

14064 UL_GETFSIZE Get maximum file size.

14065 UL_SETFSIZE Set maximum file size.

14066 The following shall be declared as a function and may also be defined as a macro. A function
14067 prototype shall be provided.

14068 long ulimit(int, ...);

14069 **APPLICATION USAGE**

14070 None.

14071 **RATIONALE**

14072 None.

14073 **FUTURE DIRECTIONS**

14074 None.

14075 **SEE ALSO**

14076 The System Interfaces volume of IEEE Std 1003.1-2001, *ulimit()*

14077 **CHANGE HISTORY**

14078 First released in Issue 3.

14079 **NAME**

14080 unistd.h — standard symbolic constants and types

14081 **SYNOPSIS**

14082 #include <unistd.h>

14083 **DESCRIPTION**

14084 The <unistd.h> header defines miscellaneous symbolic constants and types, and declares
 14085 miscellaneous functions. The actual values of the constants are unspecified except as shown. The
 14086 contents of this header are shown below.

14087 **Version Test Macros**

14088 The following symbolic constants shall be defined:

14089 _POSIX_VERSION

14090 Integer value indicating version of IEEE Std 1003.1 (C-language binding) to which the
 14091 implementation conforms. For implementations conforming to IEEE Std 1003.1-2001, the
 14092 value shall be 200112L.

14093 _POSIX2_VERSION

14094 Integer value indicating version of the Shell and Utilities volume of IEEE Std 1003.1 to
 14095 which the implementation conforms. For implementations conforming to
 14096 IEEE Std 1003.1-2001, the value shall be 200112L.

14097 The following symbolic constant shall be defined only if the implementation supports the XSI
 14098 option; see Section 2.1.4 (on page 21).

14099 XSI _XOPEN_VERSION

14100 Integer value indicating version of the X/Open Portability Guide to which the
 14101 implementation conforms. The value shall be 600.

14102 **Constants for Options and Option Groups**

14103 The following symbolic constants, if defined in <unistd.h>, shall have a value of −1, 0, or greater,
 14104 unless otherwise specified below. If these are undefined, the *fpathconf()*, *pathconf()*, or *sysconf()*
 14105 functions can be used to determine whether the option is provided for a particular invocation of
 14106 the application.

14107 If a symbolic constant is defined with the value −1, the option is not supported. Headers, data
 14108 types, and function interfaces required only for the option need not be supplied. An application
 14109 that attempts to use anything associated only with the option is considered to be requiring an
 14110 extension.

14111 If a symbolic constant is defined with a value greater than zero, the option shall always be
 14112 supported when the application is executed. All headers, data types, and functions shall be
 14113 present and shall operate as specified.

14114 If a symbolic constant is defined with the value zero, all headers, data types, and functions shall
 14115 be present. The application can check at runtime to see whether the option is supported by
 14116 calling *fpathconf()*, *pathconf()*, or *sysconf()* with the indicated *name* parameter.

14117 Unless explicitly specified otherwise, the behavior of functions associated with an unsupported
 14118 option is unspecified, and an application that uses such functions without first checking
 14119 *fpathconf()*, *pathconf()*, or *sysconf()* is considered to be requiring an extension.

14120 For conformance requirements, refer to Chapter 2 (on page 17).

14121	ADV	_POSIX_ADVISORY_INFO
14122		The implementation supports the Advisory Information option. If this symbol has a value
14123		other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14124	AIO	_POSIX_ASYNCHRONOUS_IO
14125		The implementation supports the Asynchronous Input and Output option. If this symbol
14126		has a value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14127	BAR	_POSIX_BARRIERS
14128		The implementation supports the Barriers option. If this symbol has a value other than <code>-1</code> or
14129		<code>0</code> , it shall have the value <code>200112L</code> .
14130		_POSIX_CHOWN_RESTRICTED
14131		The use of <i>chown()</i> and <i>fchown()</i> is restricted to a process with appropriate privileges, and
14132		to changing the group ID of a file only to the effective group ID of the process or to one of
14133		its supplementary group IDs. This symbol shall always be set to a value other than <code>-1</code> .
14134	CS	_POSIX_CLOCK_SELECTION
14135		The implementation supports the Clock Selection option. If this symbol has a value other
14136		than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14137	CPT	_POSIX_CPUTIME
14138		The implementation supports the Process CPU-Time Clocks option. If this symbol has a
14139		value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14140	FSC	_POSIX_FSYNC
14141		The implementation supports the File Synchronization option. If this symbol has a value
14142		other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14143		_POSIX_IPV6
14144		The implementation supports the IPv6 option. If this symbol has a value other than <code>-1</code> or <code>0</code> ,
14145		it shall have the value <code>200112L</code> .
14146		_POSIX_JOB_CONTROL
14147		The implementation supports job control. This symbol shall always be set to a value greater
14148		than zero.
14149	MF	_POSIX_MAPPED_FILES
14150		The implementation supports the Memory Mapped Files option. If this symbol has a value
14151		other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14152	ML	_POSIX_MEMLOCK
14153		The implementation supports the Process Memory Locking option. If this symbol has a
14154		value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14155	MLR	_POSIX_MEMLOCK_RANGE
14156		The implementation supports the Range Memory Locking option. If this symbol has a value
14157		other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14158	MPR	_POSIX_MEMORY_PROTECTION
14159		The implementation supports the Memory Protection option. If this symbol has a value
14160		other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14161	MSG	_POSIX_MESSAGE_PASSING
14162		The implementation supports the Message Passing option. If this symbol has a value other
14163		than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14164	MON	_POSIX_MONOTONIC_CLOCK
14165		The implementation supports the Monotonic Clock option. If this symbol has a value other

14166	than -1 or 0, it shall have the value 200112L.
14167	_POSIX_NO_TRUNC
14168	Pathname components longer than {NAME_MAX} generate an error. This symbol shall
14169	always be set to a value other than -1.
14170	_POSIX_PRIORITIZED_IO
14171	The implementation supports the Prioritized Input and Output option. If this symbol has a
14172	value other than -1 or 0, it shall have the value 200112L.
14173	_POSIX_PRIORITY_SCHEDULING
14174	The implementation supports the Process Scheduling option. If this symbol has a value
14175	other than -1 or 0, it shall have the value 200112L.
14176	_POSIX_RAW_SOCKETS
14177	The implementation supports the Raw Sockets option. If this symbol has a value other than
14178	-1 or 0, it shall have the value 200112L.
14179	_POSIX_READER_WRITER_LOCKS
14180	The implementation supports the Read-Write Locks option. This is always set to a value
14181	greater than zero if the Threads option is supported. If this symbol has a value other than -1
14182	or 0, it shall have the value 200112L.
14183	_POSIX_REALTIME_SIGNALS
14184	The implementation supports the Realtime Signals Extension option. If this symbol has a
14185	value other than -1 or 0, it shall have the value 200112L.
14186	_POSIX_REGEX
14187	The implementation supports the Regular Expression Handling option. This symbol shall
14188	always be set to a value greater than zero.
14189	_POSIX_SAVED_IDS
14190	Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always
14191	be set to a value greater than zero.
14192	_POSIX_SEMAPHORES
14193	The implementation supports the Semaphores option. If this symbol has a value other than
14194	-1 or 0, it shall have the value 200112L.
14195	_POSIX_SHARED_MEMORY_OBJECTS
14196	The implementation supports the Shared Memory Objects option. If this symbol has a value
14197	other than -1 or 0, it shall have the value 200112L.
14198	_POSIX_SHELL
14199	The implementation supports the POSIX shell. This symbol shall always be set to a value
14200	greater than zero.
14201	_POSIX_SPAWN
14202	The implementation supports the Spawn option. If this symbol has a value other than -1 or
14203	0, it shall have the value 200112L.
14204	_POSIX_SPIN_LOCKS
14205	The implementation supports the Spin Locks option. If this symbol has a value other than
14206	-1 or 0, it shall have the value 200112L.
14207	_POSIX_SPORADIC_SERVER
14208	The implementation supports the Process Sporadic Server option. If this symbol has a value
14209	other than -1 or 0, it shall have the value 200112L.

14210	SIO	_POSIX_SYNCHRONIZED_IO
14211		The implementation supports the Synchronized Input and Output option. If this symbol
14212		has a value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14213	TSA	_POSIX_THREAD_ATTR_STACKADDR
14214		The implementation supports the Thread Stack Address Attribute option. If this symbol
14215		has a value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14216	TSS	_POSIX_THREAD_ATTR_STACKSIZE
14217		The implementation supports the Thread Stack Size Attribute option. If this symbol has a
14218		value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14219	TCT	_POSIX_THREAD_CPU_TIME
14220		The implementation supports the Thread CPU-Time Clocks option. If this symbol has a
14221		value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14222	TPI	_POSIX_THREAD_PRIO_INHERIT
14223		The implementation supports the Thread Priority Inheritance option. If this symbol has a
14224		value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14225	TPP	_POSIX_THREAD_PRIO_PROTECT
14226		The implementation supports the Thread Priority Protection option. If this symbol has a
14227		value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14228	TPS	_POSIX_THREAD_PRIORITY_SCHEDULING
14229		The implementation supports the Thread Execution Scheduling option. If this symbol has a
14230		value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14231	TSH	_POSIX_THREAD_PROCESS_SHARED
14232		The implementation supports the Thread Process-Shared Synchronization option. If this
14233		symbol has a value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14234	TSF	_POSIX_THREAD_SAFE_FUNCTIONS
14235		The implementation supports the Thread-Safe Functions option. If this symbol has a value
14236		other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14237	TSP	_POSIX_THREAD_SPORADIC_SERVER
14238		The implementation supports the Thread Sporadic Server option. If this symbol has a value
14239		other than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .
14240	THR	_POSIX_THREADS
14241		The implementation supports the Threads option. If this symbol has a value other than <code>-1</code>
14242		or <code>0</code> , it shall have the value <code>200112L</code> .
14243	TMO	_POSIX_TIMEOUTS
14244		The implementation supports the Timeouts option. If this symbol has a value other than <code>-1</code>
14245		or <code>0</code> , it shall have the value <code>200112L</code> .
14246	TMR	_POSIX_TIMERS
14247		The implementation supports the Timers option. If this symbol has a value other than <code>-1</code> or
14248		<code>0</code> , it shall have the value <code>200112L</code> .
14249	TRC	_POSIX_TRACE
14250		The implementation supports the Trace option. If this symbol has a value other than <code>-1</code> or <code>0</code> ,
14251		it shall have the value <code>200112L</code> .
14252	TEF	_POSIX_TRACE_EVENT_FILTER
14253		The implementation supports the Trace Event Filter option. If this symbol has a value other
14254		than <code>-1</code> or <code>0</code> , it shall have the value <code>200112L</code> .

14255 TRI	_POSIX_TRACE_INHERIT
14256	The implementation supports the Trace Inherit option. If this symbol has a value other than
14257	−1 or 0, it shall have the value 200112L.
14258 TRL	_POSIX_TRACE_LOG
14259	The implementation supports the Trace Log option. If this symbol has a value other than −1
14260	or 0, it shall have the value 200112L.
14261 TYM	_POSIX_TYPED_MEMORY_OBJECTS
14262	The implementation supports the Typed Memory Objects option. If this symbol has a value
14263	other than −1 or 0, it shall have the value 200112L.
14264	_POSIX_VDISABLE
14265	This symbol shall be defined to be the value of a character that shall disable terminal special
14266	character handling as described in <termios.h>. This symbol shall always be set to a value
14267	other than −1.
14268	_POSIX2_C_BIND
14269	The implementation supports the C-Language Binding option. This symbol shall always
14270	have the value 200112L.
14271 CD	_POSIX2_C_DEV
14272	The implementation supports the C-Language Development Utilities option. If this symbol
14273	has a value other than −1 or 0, it shall have the value 200112L.
14274	_POSIX2_CHAR_TERM
14275	The implementation supports at least one terminal type.
14276 FD	_POSIX2_FORT_DEV
14277	The implementation supports the FORTRAN Development Utilities option. If this symbol
14278	has a value other than −1 or 0, it shall have the value 200112L.
14279 FR	_POSIX2_FORT_RUN
14280	The implementation supports the FORTRAN Runtime Utilities option. If this symbol has a
14281	value other than −1 or 0, it shall have the value 200112L.
14282	_POSIX2_LOCALEDEF
14283	The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol
14284	has a value other than −1 or 0, it shall have the value 200112L.
14285 BE	_POSIX2_PBS
14286	The implementation supports the Batch Environment Services and Utilities option. If this
14287	symbol has a value other than −1 or 0, it shall have the value 200112L.
14288 BE	_POSIX2_PBS_ACCOUNTING
14289	The implementation supports the Batch Accounting option. If this symbol has a value other
14290	than −1 or 0, it shall have the value 200112L.
14291 BE	_POSIX2_PBS_CHECKPOINT
14292	The implementation supports the Batch Checkpoint/Restart option. If this symbol has a
14293	value other than −1 or 0, it shall have the value 200112L.
14294 BE	_POSIX2_PBS_LOCATE
14295	The implementation supports the Locate Batch Job Request option. If this symbol has a
14296	value other than −1 or 0, it shall have the value 200112L.
14297 BE	_POSIX2_PBS_MESSAGE
14298	The implementation supports the Batch Job Message Request option. If this symbol has a
14299	value other than −1 or 0, it shall have the value 200112L.

14300	BE	_POSIX2_PBS_TRACK
14301		The implementation supports the Track Batch Job Request option. If this symbol has a value
14302		other than -1 or 0 , it shall have the value 200112L .
14303	SD	_POSIX2_SW_DEV
14304		The implementation supports the Software Development Utilities option. If this symbol has
14305		a value other than -1 or 0 , it shall have the value 200112L .
14306	UP	_POSIX2_UPE
14307		The implementation supports the User Portability Utilities option. If this symbol has a value
14308		other than -1 or 0 , it shall have the value 200112L .
14309		_V6_ILP32_OFF32
14310		The implementation provides a C-language compilation environment with 32-bit int , long ,
14311		pointer , and off_t types.
14312		_V6_ILP32_OFFBIG
14313		The implementation provides a C-language compilation environment with 32-bit int , long ,
14314		and pointer types and an off_t type using at least 64 bits.
14315		_V6_LP64_OFF64
14316		The implementation provides a C-language compilation environment with 32-bit int and
14317		64-bit long , pointer , and off_t types.
14318		_V6_LPBIG_OFFBIG
14319		The implementation provides a C-language compilation environment with an int type
14320		using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14321	XSI	_XBS5_ILP32_OFF32 (LEGACY)
14322		The implementation provides a C-language compilation environment with 32-bit int , long ,
14323		pointer , and off_t types.
14324	XSI	_XBS5_ILP32_OFFBIG (LEGACY)
14325		The implementation provides a C-language compilation environment with 32-bit int , long ,
14326		and pointer types and an off_t type using at least 64 bits.
14327	XSI	_XBS5_LP64_OFF64 (LEGACY)
14328		The implementation provides a C-language compilation environment with 32-bit int and
14329		64-bit long , pointer , and off_t types.
14330	XSI	_XBS5_LPBIG_OFFBIG (LEGACY)
14331		The implementation provides a C-language compilation environment with an int type
14332		using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14333	XSI	_XOPEN_CRYPT
14334		The implementation supports the X/Open Encryption Option Group.
14335		_XOPEN_ENH_I18N
14336		The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option
14337		Group. This symbol shall always be set to a value other than -1 .
14338		_XOPEN_LEGACY
14339		The implementation supports the Legacy Option Group.
14340		_XOPEN_REALTIME
14341		The implementation supports the X/Open Realtime Option Group.
14342		_XOPEN_REALTIME_THREADS
14343		The implementation supports the X/Open Realtime Threads Option Group.

14344	_XOPEN_SHM
14345	The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This
14346	symbol shall always be set to a value other than <code>-1</code> .
14347	_XOPEN_STREAMS
14348	The implementation supports the XSI STREAMS Option Group.
14349 XSI	_XOPEN_UNIX
14350	The implementation supports the XSI extension.
14351	Execution-Time Symbolic Constants
14352	If any of the following constants are not defined in the <unistd.h> header, the value shall vary
14353	depending on the file to which it is applied.
14354	If any of the following constants are defined to have value <code>-1</code> in the <unistd.h> header, the
14355	implementation shall not provide the option on any file; if any are defined to have a value other
14356	than <code>-1</code> in the <unistd.h> header, the implementation shall provide the option on all applicable
14357	files.
14358	All of the following constants, whether defined in <unistd.h> or not, may be queried with
14359	respect to a specific file using the <i>pathconf()</i> or <i>fpathconf()</i> functions:
14360	_POSIX_ASYNC_IO
14361	Asynchronous input or output operations may be performed for the associated file.
14362	_POSIX_PRIO_IO
14363	Prioritized input or output operations may be performed for the associated file.
14364	_POSIX_SYNC_IO
14365	Synchronized input or output operations may be performed for the associated file.
14366	Constants for Functions
14367	The following symbolic constant shall be defined:
14368	NULL Null pointer
14369	The following symbolic constants shall be defined for the <i>access()</i> function:
14370	F_OK Test for existence of file.
14371	R_OK Test for read permission.
14372	W_OK Test for write permission.
14373	X_OK Test for execute (search) permission.
14374	The constants F_OK , R_OK , W_OK , and X_OK and the expressions <i>R_OK</i> <i>W_OK</i> , <i>R_OK</i> <i>X_OK</i> ,
14375	and <i>R_OK</i> <i>W_OK</i> <i>X_OK</i> shall all have distinct values.
14376	The following symbolic constants shall be defined for the <i>confstr()</i> function:
14377	_CS_PATH
14378	This is the value for the <i>PATH</i> environment variable that finds all standard utilities.
14379	_CS_POSIX_V6_ILP32_OFF32_CFLAGS
14380	If <i>sysconf(_SC_V6_ILP32_OFF32)</i> returns <code>-1</code> , the meaning of this value is unspecified.
14381	Otherwise, this value is the set of initial options to be given to the <i>c99</i> utility to build an
14382	application using a programming model with 32-bit int , long , pointer , and off_t types.

14383 `_CS_POSIX_V6_ILP32_OFF32_LDFLAGS`
 14384 If `sysconf(_SC_V6_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
 14385 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
 14386 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14387 `_CS_POSIX_V6_ILP32_OFF32_LIBS`
 14388 If `sysconf(_SC_V6_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
 14389 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
 14390 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14391 `_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS`
 14392 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14393 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
 14394 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14395 **off_t** type using at least 64 bits.

14396 `_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS`
 14397 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14398 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
 14399 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14400 **off_t** type using at least 64 bits.

14401 `_CS_POSIX_V6_ILP32_OFFBIG_LIBS`
 14402 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14403 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
 14404 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14405 **off_t** type using at least 64 bits.

14406 `_CS_POSIX_V6_LP64_OFF64_CFLAGS`
 14407 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14408 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
 14409 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14410 types.

14411 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`
 14412 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14413 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
 14414 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14415 types.

14416 `_CS_POSIX_V6_LP64_OFF64_LIBS`
 14417 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14418 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
 14419 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14420 types.

14421 `_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS`
 14422 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14423 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
 14424 application using a programming model with an **int** type using at least 32 bits and **long**,
 14425 **pointer**, and **off_t** types using at least 64 bits.

14426 `_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS`
 14427 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14428 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
 14429 application using a programming model with an **int** type using at least 32 bits and **long**,
 14430 **pointer**, and **off_t** types using at least 64 bits.

14431 `_CS_POSIX_V6_LPBIG_OFFBIG_LIBS`
 14432 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14433 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
 14434 application using a programming model with an **int** type using at least 32 bits and **long**,
 14435 **pointer**, and **off_t** types using at least 64 bits.

14436 `_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS`
 14437 This value is a <newline>-separated list of names of programming environments supported
 14438 by the implementation in which the widths of the **blksize_t**, **cc_t**, **mode_t**, **nfds_t**, **pid_t**,
 14439 **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **useconds_t**, **wchar_t**, and **wint_t**
 14440 types are no greater than the width of type **long**.

14441 XSI The following symbolic constants are reserved for compatibility with Issue 5:

14442 `_CS_XBS5_ILP32_OFF32_CFLAGS (LEGACY)`
 14443 `_CS_XBS5_ILP32_OFF32_LDFLAGS (LEGACY)`
 14444 `_CS_XBS5_ILP32_OFF32_LIBS (LEGACY)`
 14445 `_CS_XBS5_ILP32_OFF32_LINTFLAGS (LEGACY)`
 14446 `_CS_XBS5_ILP32_OFFBIG_CFLAGS (LEGACY)`
 14447 `_CS_XBS5_ILP32_OFFBIG_LDFLAGS (LEGACY)`
 14448 `_CS_XBS5_ILP32_OFFBIG_LIBS (LEGACY)`
 14449 `_CS_XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY)`
 14450 `_CS_XBS5_LP64_OFF64_CFLAGS (LEGACY)`
 14451 `_CS_XBS5_LP64_OFF64_LDFLAGS (LEGACY)`
 14452 `_CS_XBS5_LP64_OFF64_LIBS (LEGACY)`
 14453 `_CS_XBS5_LP64_OFF64_LINTFLAGS (LEGACY)`
 14454 `_CS_XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)`
 14455 `_CS_XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)`
 14456 `_CS_XBS5_LPBIG_OFFBIG_LIBS (LEGACY)`
 14457 `_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY)`
 14458

14459 The following symbolic constants shall be defined for the `lseek()` and `fcntl()` functions and shall
 14460 have distinct values:

14461 `SEEK_CUR` Set file offset to current plus *offset*.
 14462 `SEEK_END` Set file offset to EOF plus *offset*.
 14463 `SEEK_SET` Set file offset to *offset*.

14464 The following symbolic constants shall be defined as possible values for the *function* argument
 14465 to the `lockf()` function:

14466 `F_LOCK` Lock a section for exclusive use.
 14467 `F_TEST` Test section for locks by other processes.
 14468 `F_TLOCK` Test and lock a section for exclusive use.
 14469 `F_ULOCK` Unlock locked sections.

14470 The following symbolic constants shall be defined for `pathconf()`:

14471 `_PC_ALLOC_SIZE_MIN`
 14472 `_PC_ASYNC_IO`
 14473 `_PC_CHOWN_RESTRICTED`
 14474 `_PC_FILESIZEBITS`
 14475 `_PC_LINK_MAX`

```

14476     _PC_MAX_CANON
14477     _PC_MAX_INPUT
14478     _PC_NAME_MAX
14479     _PC_NO_TRUNC
14480     _PC_PATH_MAX
14481     _PC_PIPE_BUF
14482     _PC_PRIO_IO
14483     _PC_REC_INCR_XFER_SIZE
14484     _PC_REC_MIN_XFER_SIZE
14485     _PC_REC_XFER_ALIGN
14486     _PC_SYMLINK_MAX
14487     _PC_SYNC_IO
14488     _PC_VDISABLE

```

14489 The following symbolic constants shall be defined for *sysconf()*:

```

14490     _SC_2_C_BIND
14491     _SC_2_C_DEV
14492     _SC_2_C_VERSION
14493     _SC_2_CHAR_TERM
14494     _SC_2_FORT_DEV
14495     _SC_2_FORT_RUN
14496     _SC_2_LOCALEDEF
14497     _SC_2_PBS
14498     _SC_2_PBS_ACCOUNTING
14499     _SC_2_PBS_CHECKPOINT
14500     _SC_2_PBS_LOCATE
14501     _SC_2_PBS_MESSAGE
14502     _SC_2_PBS_TRACK
14503     _SC_2_SW_DEV
14504     _SC_2_UPE
14505     _SC_2_VERSION
14506     _SC_ADVISORY_INFO
14507     _SC_ARG_MAX
14508     _SC_AIO_LISTIO_MAX
14509     _SC_AIO_MAX
14510     _SC_AIO_PRIO_DELTA_MAX
14511     _SC_ASYNCHRONOUS_IO
14512     _SC_ATEXIT_MAX
14513     _SC_BARRIERS
14514     _SC_BC_BASE_MAX
14515     _SC_BC_DIM_MAX
14516     _SC_BC_SCALE_MAX
14517     _SC_BC_STRING_MAX
14518     _SC_CHILD_MAX
14519     _SC_CLK_TCK
14520     _SC_CLOCK_SELECTION
14521     _SC_COLL_WEIGHTS_MAX
14522     _SC_CPUTIME
14523     _SC_DELAYTIMER_MAX
14524     _SC_EXPR_NEST_MAX
14525     _SC_FILE_LOCKING
14526     _SC_FSYNC

```

```

14527     _SC_GETGR_R_SIZE_MAX
14528     _SC_GETPW_R_SIZE_MAX
14529     _SC_HOST_NAME_MAX
14530     _SC_IOV_MAX
14531     _SC_IPV6
14532     _SC_JOB_CONTROL
14533     _SC_LINE_MAX
14534     _SC_LOGIN_NAME_MAX
14535     _SC_MAPPED_FILES
14536     _SC_MEMLOCK
14537     _SC_MEMLOCK_RANGE
14538     _SC_MEMORY_PROTECTION
14539     _SC_MESSAGE_PASSING
14540     _SC_MONOTONIC_CLOCK
14541     _SC_MQ_OPEN_MAX
14542     _SC_MQ_PRIO_MAX
14543     _SC_NGROUPS_MAX
14544     _SC_OPEN_MAX
14545     _SC_PAGE_SIZE
14546     _SC_PAGESIZE
14547     _SC_PRIORITIZED_IO
14548     _SC_PRIORITY_SCHEDULING
14549     _SC_RAW_SOCKETS
14550     _SC_RE_DUP_MAX
14551     _SC_READER_WRITER_LOCKS
14552     _SC_REALTIME_SIGNALS
14553     _SC_REGEX
14554     _SC_RTSIG_MAX
14555     _SC_SAVED_IDS
14556     _SC_SEMAPHORES
14557     _SC_SEM_NSEMS_MAX
14558     _SC_SEM_VALUE_MAX
14559     _SC_SHARED_MEMORY_OBJECTS
14560     _SC_SHELL
14561     _SC_SIGQUEUE_MAX
14562     _SC_SPAWN
14563     _SC_SPIN_LOCKS
14564     _SC_SPORADIC_SERVER
14565     _SC_STREAM_MAX
14566     _SC_SYMLOOP_MAX
14567     _SC_SYNCHRONIZED_IO
14568     _SC_THREAD_ATTR_STACKADDR
14569     _SC_THREAD_ATTR_STACKSIZE
14570     _SC_THREAD_CPUTIME
14571     _SC_THREAD_DESTRUCTOR_ITERATIONS
14572     _SC_THREAD_KEYS_MAX
14573     _SC_THREAD_PRIO_INHERIT
14574     _SC_THREAD_PRIO_PROTECT
14575     _SC_THREAD_PRIORITY_SCHEDULING
14576     _SC_THREAD_PROCESS_SHARED
14577     _SC_THREAD_SAFE_FUNCTIONS
14578     _SC_THREAD_SPORADIC_SERVER

```



```

14579     _SC_THREAD_STACK_MIN
14580     _SC_THREAD_THREADS_MAX
14581     _SC_TIMEOUTS
14582     _SC_THREADS
14583     _SC_TIMER_MAX
14584     _SC_TIMERS
14585     _SC_TRACE
14586     _SC_TRACE_EVENT_FILTER
14587     _SC_TRACE_INHERIT
14588     _SC_TRACE_LOG
14589     _SC_TTY_NAME_MAX
14590     _SC_TYPED_MEMORY_OBJECTS
14591     _SC_TZNAME_MAX
14592     _SC_V6_ILP32_OFF32
14593     _SC_V6_ILP32_OFFBIG
14594     _SC_V6_LP64_OFF64
14595     _SC_V6_LPBIG_OFFBIG
14596     _SC_VERSION
14597     _SC_XBS5_ILP32_OFF32 (LEGACY)
14598     _SC_XBS5_ILP32_OFFBIG (LEGACY)
14599     _SC_XBS5_LP64_OFF64 (LEGACY)
14600     _SC_XBS5_LPBIG_OFFBIG (LEGACY)
14601     _SC_XOPEN_CRYPT
14602     _SC_XOPEN_ENH_I18N
14603     _SC_XOPEN_LEGACY
14604     _SC_XOPEN_REALTIME
14605     _SC_XOPEN_REALTIME_THREADS
14606     _SC_XOPEN_SHM
14607     _SC_XOPEN_STREAMS
14608     _SC_XOPEN_UNIX
14609     _SC_XOPEN_VERSION
14610     _SC_XOPEN_XCU_VERSION

14611     The two constants _SC_PAGESIZE and _SC_PAGE_SIZE may be defined to have the same |
14612     value.

14613     The following symbolic constants shall be defined for file streams:

14614     STDERR_FILENO      File number of stderr; 2.
14615     STDIN_FILENO       File number of stdin; 0.
14616     STDOUT_FILENO      File number of stdout; 1.

14617     Type Definitions

14618     The size_t, ssize_t, uid_t, gid_t, off_t, pid_t, and useconds_t types shall be defined as described
14619     in <sys/types.h>.

14620     The intptr_t type shall be defined as described in <inttypes.h>.

```

14621 **Declarations**

14622 The following shall be declared as functions and may also be defined as macros. Function
 14623 prototypes shall be provided.

```

14624      int            access(const char *, int);
14625      unsigned       alarm(unsigned);
14626      int            chdir(const char *);
14627      int            chown(const char *, uid_t, gid_t);
14628      int            close(int);
14629      size_t         confstr(int, char *, size_t);
14630 XSI      char        *crypt(const char *, const char *);
14631      char        *ctermid(char *);
14632      int            dup(int);
14633      int            dup2(int, int);
14634 XSI      void        encrypt(char[64], int);
14635      int            execl(const char *, const char *, ...);
14636      int            execle(const char *, const char *, ...);
14637      int            execlp(const char *, const char *, ...);
14638      int            execv(const char *, char *const []);
14639      int            execve(const char *, char *const [], char *const []);
14640      int            execvp(const char *, char *const []);
14641      void           _exit(int);
14642      int            fchown(int, uid_t, gid_t);
14643 XSI      int           fchdir(int);
14644 SIO      int           fdatsync(int);
14645      pid_t          fork(void);
14646      long           fpathconf(int, int);
14647 FSC      int           fsync(int);
14648      int            ftruncate(int, off_t);
14649      char           *getcwd(char *, size_t);
14650      gid_t           getegid(void);
14651      uid_t           geteuid(void);
14652      gid_t           getgid(void);
14653      int            getgroups(int, gid_t []);
14654 XSI      long          gethostid(void);
14655      int            gethostname(char *, size_t);
14656      char           *getlogin(void);
14657      int            getlogin_r(char *, size_t);
14658      int            getopt(int, char * const [], const char *);
14659 XSI      pid_t          getpgid(pid_t);
14660      pid_t          getpgrp(void);
14661      pid_t          getpid(void);
14662      pid_t          getppid(void);
14663 XSI      pid_t          getsid(pid_t);
14664      uid_t           getuid(void);
14665 XSI      char           *getwd(char *); (LEGACY)
14666      int            isatty(int);
14667 XSI      int           lchown(const char *, uid_t, gid_t);
14668      int            link(const char *, const char *);
14669 XSI      int           lockf(int, int, off_t);
14670      off_t          lseek(int, off_t, int);
14671 XSI

```

```

14672     int             nice(int);
14673     long            pathconf(const char *, int);
14674     int             pause(void);
14675     int             pipe(int [2]);
14676 XSI    ssize_t      pread(int, void *, size_t, off_t);
14677     ssize_t         pwrite(int, const void *, size_t, off_t);
14678     ssize_t         read(int, void *, size_t);
14679     ssize_t         readlink(const char *restrict, char *restrict, size_t);
14680     int             rmdir(const char *);
14681     int             setegid(gid_t);
14682     int             seteuid(uid_t);
14683     int             setgid(gid_t);
14684     int             setpgid(pid_t, pid_t);
14685 XSI    pid_t        setpgrp(void);
14686     int             setregid(gid_t, gid_t);
14687     int             setreuid(uid_t, uid_t);
14688     pid_t           setsid(void);
14689     int             setuid(uid_t);
14690     unsigned        sleep(unsigned);
14691 XSI    void          swab(const void *restrict, void *restrict, ssize_t);
14692     int             symlink(const char *, const char *);
14693     void            sync(void);
14694     long            sysconf(int);
14695     pid_t           tcgetpgrp(int);
14696     int             tcsetpgrp(int, pid_t);
14697 XSI    int           truncate(const char *, off_t);
14698     char            *ttyname(int);
14699     int             ttyname_r(int, char *, size_t);
14700 XSI    useconds_t    ualarm(useconds_t, useconds_t);
14701     int             unlink(const char *);
14702 XSI    int           usleep(useconds_t);
14703     pid_t           vfork(void);
14704     ssize_t         write(int, const void *, size_t);

```

14705 Implementations may also include the *pthread_atfork()* prototype as defined in <pthread.h> (on
14706 page 289).

14707 The following external variables shall be declared:

```

14708     extern char      *optarg;
14709     extern int       optind, opterr, optopt;

```

14710 APPLICATION USAGE

14711 IEEE Std 1003.1-2001 only describes the behavior of systems that claim conformance to it.
14712 However, application developers who want to write applications that adapt to other versions of
14713 IEEE Std 1003.1 (or to systems that do not conform to any POSIX standard) may find it useful to
14714 code them so as to conditionally compile different code depending on the value of
14715 `_POSIX_VERSION`, for example:

```

14716     #if _POSIX_VERSION >= 200112L
14717     /* Use the newer function that copes with large files. */
14718     off_t pos=ftello(fp);
14719     #else
14720     /* Either this is an old version of POSIX, or _POSIX_VERSION is
14721        not even defined, so use the traditional function. */

```

```

14722     long pos=ftell(fp);
14723     #endif

14724     Earlier versions of IEEE Std 1003.1 and of the Single UNIX Specification can be identified by the
14725     following macros:

14726     POSIX.1-1988 standard
14727         _POSIX_VERSION==198808L

14728     POSIX.1-1990 standard
14729         _POSIX_VERSION==199009L

14730     ISO POSIX-1: 1996 standard
14731         _POSIX_VERSION==199506L

14732     Single UNIX Specification, Version 1
14733         _XOPEN_UNIX and _XOPEN_VERSION==4

14734     Single UNIX Specification, Version 2
14735         _XOPEN_UNIX and _XOPEN_VERSION==500

14736     IEEE Std 1003.1-2001 does not make any attempt to define application binary interaction with
14737     the underlying operating system. However, application developers may find it useful to query
14738     _SC_VERSION at runtime via sysconf() to determine whether the current version of the
14739     operating system supports the necessary functionality as in the following program fragment:

14740     if (sysconf(_SC_VERSION) < 200112L) {
14741         fprintf(stderr, "POSIX.1-2001 system required, terminating \n");
14742         exit(1);
14743     }

14744     New applications should not use _XOPEN_SHM or _XOPEN_ENH_I18N.

```

14745 **RATIONALE**

14746 As IEEE Std 1003.1-2001 evolved, certain options became sufficiently standardized that it was
 14747 concluded that simply requiring one of the option choices was simpler than retaining the option.
 14748 However, for backwards-compatibility, the option flags (with required constant values) are
 14749 retained.

14750 **Version Test Macros**

14751 The standard developers considered altering the definition of `_POSIX_VERSION` and removing
 14752 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed
 14753 by some to be minimal, and since the implementation of the functionality is potentially
 14754 problematic. However, they recognized that support for existing application binaries is a
 14755 concern to manufacturers, application developers, and the users of implementations conforming
 14756 to IEEE Std 1003.1-2001.

14757 While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide
 14758 the greatest degree of imaginable utility to the application developer or user, it is arguably better
 14759 than a **core** file or some other equally obscure result. (It is also possible for implementations to
 14760 encode and recognize application binaries compiled in various POSIX.1-conforming
 14761 environments, and modify the semantics of the underlying system to conform to the
 14762 expectations of the application.) For the reasons outlined in the preceding paragraphs and in the
 14763 APPLICATION USAGE section, the standard developers elected to retain the `_POSIX_VERSION`
 14764 and `_SC_VERSION` functionality.

Compile-Time Symbolic Constants for System-Wide Options

IEEE Std 1003.1-2001 now includes support in certain areas for the newly adopted policy governing options and stubs.

This policy provides flexibility for implementations in how they support options. It also specifies how conforming applications can adapt to different implementations that support different sets of options. It allows the following:

1. If an implementation has no interest in supporting an option, it does not have to provide anything associated with that option beyond the announcement that it does not support it.
2. An implementation can support a partial or incompatible version of an option (as a non-standard extension) as long as it does not claim to support the option.
3. An application can determine whether the option is supported. A strictly conforming application must check this announcement mechanism before first using anything associated with the option.

There is an important implication of this policy. IEEE Std 1003.1-2001 cannot dictate the behavior of interfaces associated with an option when the implementation does not claim to support the option. In particular, it cannot require that a function associated with an unsupported option will fail if it does not perform as specified. However, this policy does not prevent a standard from requiring certain functions to always be present, but that they shall always fail on some implementations. The *setpgid()* function in the POSIX.1-1990 standard, for example, is considered appropriate.

The POSIX standards include various options, and the C-language binding support for an option implies that the implementation must supply data types and function interfaces. An application must be able to discover whether the implementation supports each option.

Any application must consider the following three cases for each option:

1. Option never supported.

The implementation advertises at compile time that the option will never be supported. In this case, it is not necessary for the implementation to supply any of the data types or function interfaces that are provided only as part of the option. The implementation might provide data types and functions that are similar to those defined by IEEE Std 1003.1-2001, but there is no guarantee for any particular behavior.

2. Option always supported.

The implementation advertises at compile time that the option will always be supported. In this case, all data types and function interfaces shall be available and shall operate as specified.

3. Option might or might not be supported.

Some implementations might not provide a mechanism to specify support of options at compile time. In addition, the implementation might be unable or unwilling to specify support or non-support at compile time. In either case, any application that might use the option at runtime must be able to compile and execute. The implementation must provide, at compile time, all data types and function interfaces that are necessary to allow this. In this situation, there must be a mechanism that allows the application to query, at runtime, whether the option is supported. If the application attempts to use the option when it is not supported, the result is unspecified unless explicitly specified otherwise in IEEE Std 1003.1-2001.

14809 **FUTURE DIRECTIONS**

14810 None.

14811 **SEE ALSO**

14812 <inttypes.h>, <limits.h>, <sys/socket.h>, <sys/types.h>, <termios.h>, <wctype.h>, the System
 14813 Interfaces volume of IEEE Std 1003.1-2001, *access()*, *alarm()*, *chdir()*, *chown()*, *close()*, *crypt()*,
 14814 *ctermid()*, *dup()*, *encrypt()*, *environ*, *exec*, *exit()*, *fchdir()*, *fchown()*, *fcntl()*, *fork()*, *fpathconf()*,
 14815 *fsync()*, *ftruncate()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*,
 14816 *getlogin()*, *getpgid()*, *getpgrp()*, *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*,
 14817 *lockf()*, *lseek()*, *nice()*, *pathconf()*, *pause()*, *pipe()*, *read()*, *readlink()*, *rmdir()*, *setgid()*, *setpgid()*,
 14818 *setpgrp()*, *setregid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*,
 14819 *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *ualarm()*, *unlink()*, *usleep()*, *vfork()*, *write()*

14820 **CHANGE HISTORY**

14821 First released in Issue 1. Derived from Issue 1 of the SVID.

14822 **Issue 5**

14823 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 14824 Threads Extension.

14825 The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added.
 14826 `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other
 14827 than `-1` by a conforming implementation.

14828 Large File System extensions are added.

14829 The type of the argument to *sbrk()* is changed from **int** to **intptr_t**.

14830 `_XBS_` constants are added to the list of constants for Options and Option Groups, to the list of
 14831 constants for the *confstr()* function, and to the list of constants to the *sysconf()* function. These
 14832 are all marked EX.

14833 **Issue 6**14834 `_POSIX2_C_VERSION` is removed.14835 The Open Group Corrigendum U026/4 is applied, adding the prototype for *fdatasync()*.

14836 The Open Group Corrigendum U026/1 is applied, adding the symbols `_SC_XOPEN_LEGACY`,
 14837 `_SC_XOPEN_REALTIME`, and `_SC_XOPEN_REALTIME_THREADS`.

14838 The symbols `_XOPEN_STREAMS` and `_SC_XOPEN_STREAMS` are added to support the XSI
 14839 STREAMS Option Group.

14840 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in
 14841 IEEE Std 1003.1-2001.

14842 The legacy symbol `_SC_PASS_MAX` is removed.

14843 The following new requirements on POSIX implementations derive from alignment with the
 14844 Single UNIX Specification:

- 14845 • The `_CS_POSIX_*` and `_CS_XBS5_*` constants are added for the *confstr()* function.
- 14846 • The `_SC_XBS5_*` constants are added for the *sysconf()* function.
- 14847 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.
- 14848 • The **uid_t**, **gid_t**, **off_t**, **pid_t**, and **useconds_t** types are mandated.

14849 The *gethostname()* prototype is added for sockets.

14850 A new section is added for System-Wide Options.

14851 Function prototypes for *setegid()* and *seteuid()* are added.

14852 Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`,
 14853 `_POSIX_SPAWN`, `_POSIX_SPORADIC_SERVER`, `_POSIX_THREAD_CPUTIME`,
 14854 `_POSIX_THREAD_SPORADIC_SERVER`, and `_POSIX_TIMEOUTS`, and *pathconf()* variables are
 14855 added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`,
 14856 `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with
 14857 IEEE Std 1003.1d-1999.

14858 The following are added for alignment with IEEE Std 1003.1j-2000:

- 14859 • Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`,
 14860 `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`,
 14861 `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`
- 14862 • *sysconf()* variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,
 14863 `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and
 14864 `_SC_TYPED_MEMORY_OBJECTS`

14865 The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,
 14866 and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are
 14867 introduced.

14868 The *getwd()* function is marked LEGACY.

14869 The **restrict** keyword is added to the prototypes for *readlink()* and *swab()*.

14870 Constants for options are now harmonized, so when supported they take the year of approval of
 14871 IEEE Std 1003.1-2001 as the value.

14872 The following are added for alignment with IEEE Std 1003.1q-2000:

- 14873 • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,
 14874 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`
- 14875 • The *sysconf()* symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,
 14876 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`

14877 The *brk()* and *sbrk()* legacy functions are removed.

14878 The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning
 14879 information.

14880 The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter for
 14881 *gethostname()* from **socklen_t** to **size_t**.

14882 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/2 is applied, changing “Thread Stack
 14883 Address Size” to “Thread Stack Size Attribute”.

14884 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/20 is applied, adding the `_POSIX_IPV6`,
 14885 `_SC_V6`, and `_SC_RAW_SOCKETS` symbols.

14886 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/21 is applied, correcting the description in
 14887 “Constants for Functions” for the `_CS_POSIX_V6_LP64_OFF64_CFLAGS`,
 14888 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`, and `_CS_POSIX_V6_LP64_OFF64_LIBS` symbols.

14889 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/22 is applied, removing the shading for
 14890 the `_PC*` and `_SC*` constants, since these are mandatory on all implementations.

14891	IEEE Std 1003.1-2001 / Cor 1-2002, item XBD / TC1 / D6 / 23 is applied, adding the	
14892	_PC_SYMLINK_MAX and _SC_SYMLINK_MAX constants.	
14893	IEEE Std 1003.1-2001 / Cor 1-2002, item XBD / TC1 / D6 / 24 is applied, correcting the shading and	
14894	margin code for the <i>fsync()</i> function.	
14895	IEEE Std 1003.1-2001 / Cor 1-2002, item XBD / TC1 / D6 / 25 is applied, adding the following text to	
14896	the APPLICATION USAGE section: “New applications should not use _XOPEN_SHM or	
14897	_XOPEN_ENH_I18N.”.	

14898 **NAME**

14899 utime.h — access and modification times structure

14900 **SYNOPSIS**

14901 #include <utime.h>

14902 **DESCRIPTION**

14903 The <utime.h> header shall declare the structure **utimbuf**, which shall include the following
14904 members:

14905 time_t actime Access time.
14906 time_t modtime Modification time.

14907 The times shall be measured in seconds since the Epoch.

14908 The type **time_t** shall be defined as described in <sys/types.h>.

14909 The following shall be declared as a function and may also be defined as a macro. A function
14910 prototype shall be provided.

14911 int utime(const char *, const struct utimbuf *);

14912 **APPLICATION USAGE**

14913 None.

14914 **RATIONALE**

14915 None.

14916 **FUTURE DIRECTIONS**

14917 None.

14918 **SEE ALSO**

14919 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *utime()*

14920 **CHANGE HISTORY**

14921 First released in Issue 3.

14922 **Issue 6**

14923 The following new requirements on POSIX implementations derive from alignment with the
14924 Single UNIX Specification:

- 14925 • The **time_t** type is defined.

14926 **NAME**

14927 utmpx.h — user accounting database definitions

14928 **SYNOPSIS**14929 xSI `#include <utmpx.h>`

14930

14931 **DESCRIPTION**14932 The <utmpx.h> header shall define the **utmpx** structure that shall include at least the following
14933 members:

14934	char	ut_user[]	User login name.
14935	char	ut_id[]	Unspecified initialization process identifier.
14936	char	ut_line[]	Device name.
14937	pid_t	ut_pid	Process ID.
14938	short	ut_type	Type of entry.
14939	struct timeval	ut_tv	Time entry was made.

14940 The **pid_t** type shall be defined through **typedef** as described in <sys/types.h>.14941 The **timeval** structure shall be defined as described in <sys/time.h>.

14942 Inclusion of the <utmpx.h> header may also make visible all symbols from <sys/time.h>.

14943 The following symbolic constants shall be defined as possible values for the *ut_type* member of
14944 the **utmpx** structure:

14945	EMPTY	No valid user accounting information.
14946	BOOT_TIME	Identifies time of system boot.
14947	OLD_TIME	Identifies time when system clock changed.
14948	NEW_TIME	Identifies time after system clock changed.
14949	USER_PROCESS	Identifies a process.
14950	INIT_PROCESS	Identifies a process spawned by the init process.
14951	LOGIN_PROCESS	Identifies the session leader of a logged-in user.
14952	DEAD_PROCESS	Identifies a session leader who has exited.

14953 The following shall be declared as functions and may also be defined as macros. Function
14954 prototypes shall be provided.

```

14955 void          endutxent(void);
14956 struct utmpx *getutxent(void);
14957 struct utmpx *getutxid(const struct utmpx *);
14958 struct utmpx *getutxline(const struct utmpx *);
14959 struct utmpx *pututxline(const struct utmpx *);
14960 void          setutxent(void);

```

14961 **APPLICATION USAGE**

14962 None.

14963 **RATIONALE**

14964 None.

14965 **FUTURE DIRECTIONS**

14966 None.

14967 **SEE ALSO**

14968 <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *endutxent()*

14969 **CHANGE HISTORY**

14970 First released in Issue 4, Version 2.

14971 **NAME**

14972 wchar.h — wide-character handling

14973 **SYNOPSIS**

14974 #include <wchar.h>

14975 **DESCRIPTION**

14976 CX Some of the functionality described on this reference page extends the ISO C standard.
 14977 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 14978 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 14979 symbols in this header.

14980 The <wchar.h> header shall define the following types:

14981 **wchar_t** As described in <stddef.h>.14982 **wint_t** An integer type capable of storing any valid value of **wchar_t** or WEOF.

14983 XSI **wctype_t** A scalar type of a data object that can hold values which represent locale-
 14984 specific character classification.

14985 **mbstate_t** An object type other than an array type that can hold the conversion state
 14986 information necessary to convert between sequences of (possibly multi-byte)
 14987 XSI characters and wide characters. If a codeset is being used such that an
 14988 **mbstate_t** needs to preserve more than 2 levels of reserved state, the results
 14989 are unspecified.

14990 XSI **FILE** As described in <stdio.h>.14991 **size_t** As described in <stddef.h>.14992 XSI **va_list** As described in <stdarg.h>.

14993 The implementation shall support one or more programming environments in which the width
 14994 of **wint_t** is no greater than the width of type **long**. The names of these programming
 14995 environments can be obtained using the *confstr()* function or the *getconf* utility.

14996 The following shall be declared as functions and may also be defined as macros. Function
 14997 prototypes shall be provided.

```

14998       wint_t       btowc(int);
14999       wint_t       fgetwc(FILE *);
15000       wchar_t      *fgetws(wchar_t *restrict, int, FILE *restrict);
15001       wint_t       fputwc(wchar_t, FILE *);
15002       int          fputws(const wchar_t *restrict, FILE *restrict);
15003       int          fwide(FILE *, int);
15004       int          fwprintf(FILE *restrict, const wchar_t *restrict, ...);
15005       int          fwscanf(FILE *restrict, const wchar_t *restrict, ...);
15006       wint_t       getwc(FILE *);
15007       wint_t       getwchar(void);
15008 XSI       int       iswalnum(wint_t);
15009       int       iswalpha(wint_t);
15010       int       iswcntrl(wint_t);
15011       int       iswctype(wint_t, wctype_t);
15012       int       iswdigit(wint_t);
15013       int       iswgraph(wint_t);
15014       int       iswlower(wint_t);
15015       int       iswprint(wint_t);
15016       int       iswpunct(wint_t);

```

```

15017     int             iswspace(wint_t);
15018     int             iswupper(wint_t);
15019     int             iswxdigit(wint_t);
15020     size_t          mbrlen(const char *restrict, size_t, mbstate_t *restrict);
15021     size_t          mbrtowc(wchar_t *restrict, const char *restrict, size_t,
15022                             mbstate_t *restrict);
15023     int             mbsinit(const mbstate_t *);
15024     size_t          mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
15025                               mbstate_t *restrict);
15026     wint_t          putwc(wchar_t, FILE *);
15027     wint_t          putwchar(wchar_t);
15028     int             swprintf(wchar_t *restrict, size_t,
15029                              const wchar_t *restrict, ...);
15030     int             swscanf(const wchar_t *restrict,
15031                              const wchar_t *restrict, ...);
15032     wint_t          towlower(wint_t);
15033     wint_t          towupper(wint_t);
15034     wint_t          ungetwc(wint_t, FILE *);
15035     int             vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15036     int             vwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15037     int             vwprintf(const wchar_t *restrict, va_list);
15038     int             vswprintf(wchar_t *restrict, size_t,
15039                               const wchar_t *restrict, va_list);
15040     int             vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15041                              va_list);
15042     int             vwscanf(const wchar_t *restrict, va_list);
15043     size_t          wctomb(char *restrict, wchar_t, mbstate_t *restrict);
15044     wchar_t         *wcscat(wchar_t *restrict, const wchar_t *restrict);
15045     wchar_t         *wcschr(const wchar_t *, wchar_t);
15046     int             wcscmp(const wchar_t *, const wchar_t *);
15047     int             wcscoll(const wchar_t *, const wchar_t *);
15048     wchar_t         *wcscpy(wchar_t *restrict, const wchar_t *restrict);
15049     size_t          wcscspn(const wchar_t *, const wchar_t *);
15050     size_t          wcsftime(wchar_t *restrict, size_t,
15051                              const wchar_t *restrict, const struct tm *restrict);
15052     size_t          wcslen(const wchar_t *);
15053     wchar_t         *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15054     int             wcsncmp(const wchar_t *, const wchar_t *, size_t);
15055     wchar_t         *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15056     wchar_t         *wcspbrk(const wchar_t *, const wchar_t *);
15057     wchar_t         *wcsrchr(const wchar_t *, wchar_t);
15058     size_t          wcsrtombs(char *restrict, const wchar_t **restrict,
15059                               size_t, mbstate_t *restrict);
15060     size_t          wcsspncpy(const wchar_t *, const wchar_t *);
15061     wchar_t         *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15062     double          wcstod(const wchar_t *restrict, wchar_t **restrict);
15063     float           wcstof(const wchar_t *restrict, wchar_t **restrict);
15064     wchar_t         *wcstok(wchar_t *restrict, const wchar_t *restrict,
15065                               wchar_t **restrict);
15066     long            wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15067     long double     wcstold(const wchar_t *restrict, wchar_t **restrict);
15068     long long       wcstoll(const wchar_t *restrict, wchar_t **restrict, int);

```

```

15069 unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15070 unsigned long long
15071         wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15072 XSI wchar_t *wcswcs(const wchar_t *, const wchar_t *);
15073 int wcswidth(const wchar_t *, size_t);
15074 size_t wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15075 int wctob(wint_t);
15076 XSI wctype_t wctype(const char *);
15077 int wcwidth(wchar_t);
15078 wchar_t *wmemchr(const wchar_t *, wchar_t, size_t);
15079 int wmemcmp(const wchar_t *, const wchar_t *, size_t);
15080 wchar_t *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15081 wchar_t *wmemmove(wchar_t *, const wchar_t *, size_t);
15082 wchar_t *wmemset(wchar_t *, wchar_t, size_t);
15083 int wprintf(const wchar_t *restrict, ...);
15084 int wscanf(const wchar_t *restrict, ...);

```

15085 The <wchar.h> header shall define the following macros:

15086	WCHAR_MAX	The maximum value representable by an object of type wchar_t .
15087	WCHAR_MIN	The minimum value representable by an object of type wchar_t .
15088	WEOF	Constant expression of type wint_t that is returned by several WP functions
15089		to indicate end-of-file.
15090	NULL	As described in <stddef.h>.

15091 The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are
 15092 described in the header <time.h>.

15093 CX Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,
 15094 <string.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, and <time.h>.

15095 APPLICATION USAGE

15096 The *iswblank()* function was a late addition to the ISO C standard and was introduced at the
 15097 same time as the ISO C standard introduced <wctype.h>, which contains all of the *isw*()*
 15098 functions. The Open Group Base Specifications had previously aligned with the MSE working
 15099 draft and had introduced the rest of the *isw*()* functions into <wchar.h>. For backwards-
 15100 compatibility, the original set of *isw*()* functions, without *iswblank()*, are permitted (as an XSI
 15101 extension) in <wchar.h>. For maximum portability, applications should include <wctype.h> in
 15102 order to obtain declarations for the *isw*()* functions.

15103 RATIONALE

15104 In the ISO C standard, the symbols referenced as XSI extensions are in <wctype.h>. Their
 15105 presence here is thus an extension.

15106 FUTURE DIRECTIONS

15107 None.

15108 SEE ALSO

15109 <ctype.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, <wctype.h>, the
 15110 System Interfaces volume of IEEE Std 1003.1-2001, *btowc()*, *confstr()*, *fgetwc()*, *fgetws()*, *fputwc()*,
 15111 *fputws()*, *fwide()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalpalpha()*, *iswcntrl()*,
 15112 *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*,
 15113 *iswxdigit()*, *iswctype()*, *mbsinit()*, *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *putwc()*, *putwchar()*,
 15114 *swprintf()*, *swscanf()*, *towlower()*, *towupper()*, *ungetwc()*, *vfwprintf()*, *vfwscanf()*, *vswprintf()*,
 15115 *vswscanf()*, *vwscanf()*, *wcrtomb()*, *wcsrtombs()*, *wcscat()*, *wcschr()*, *wcscmp()*, *wcscoll()*, *wcscpy()*,

15116 *wscspn()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsspn()*,
 15117 *wcsstr()*, *wctod()*, *wcstof()*, *wctok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstoul()*, *wcstoull()*, *wcswcs()*,
 15118 *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*,
 15119 *wmemmove()*, *wmemset()*, *wprintf()*, *wscanf()*, the Shell and Utilities volume of
 15120 IEEE Std 1003.1-2001, *getconf*

15121 **CHANGE HISTORY**

15122 First released in Issue 4.

15123 **Issue 5**

15124 Aligned with the ISO / IEC 9899: 1990 / Amendment 1: 1995 (E).

15125 **Issue 6**

15126 The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and
 15127 *wcwidth()* are marked as extensions.

15128 The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()*
 15129 function.

15130 The following changes are made for alignment with the ISO / IEC 9899: 1999 standard:

- 15131 • Various function prototypes are updated to add the **restrict** keyword.
- 15132 • The functions *vwscanf()*, *vswscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are added.

15133 The type **wctype_t**, the *isw*()*, *to*()*, and *wctype()* functions are marked as XSI extensions.

15134 IEEE Std 1003.1-2001 / Cor 1-2002, item XBD / TC1 / D6 / 26 is applied, adding the APPLICATION
 15135 USAGE section.

15136 **NAME**

15137 wctype.h — wide-character classification and mapping utilities

15138 **SYNOPSIS**

15139 #include <wctype.h>

15140 **DESCRIPTION**

15141 CX Some of the functionality described on this reference page extends the ISO C standard.
 15142 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 15143 IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these
 15144 symbols in this header.

15145 The <wctype.h> header shall define the following types:

15146 **wint_t** As described in <wchar.h>.

15147 **wctrans_t** A scalar type that can hold values which represent locale-specific character
 15148 mappings.

15149 **wctype_t** As described in <wchar.h>.

15150 The following shall be declared as functions and may also be defined as macros. Function
 15151 prototypes shall be provided.

```

15152 int      iswalnum(wint_t);
15153 int      iswalpha(wint_t);
15154 int      iswblank(wint_t);
15155 int      iswcntrl(wint_t);
15156 int      iswdigit(wint_t);
15157 int      iswgraph(wint_t);
15158 int      iswlower(wint_t);
15159 int      iswprint(wint_t);
15160 int      iswpunct(wint_t);
15161 int      iswspace(wint_t);
15162 int      iswupper(wint_t);
15163 int      iswxdigit(wint_t);
15164 int      iswctype(wint_t, wctype_t);
15165 wint_t   towctrans(wint_t, wctrans_t);
15166 wint_t   tolower(wint_t);
15167 wint_t   towupper(wint_t);
15168 wctrans_t wctrans(const char *);
15169 wctype_t wctype(const char *);

```

15170 The <wctype.h> header shall define the following macro name:

15171 **WEOF** Constant expression of type **wint_t** that is returned by several MSE functions
 15172 to indicate end-of-file.

15173 For all functions described in this header that accept an argument of type **wint_t**, the value is
 15174 representable as a **wchar_t** or equals the value of WEOF. If this argument has any other value,
 15175 the behavior is undefined.

15176 The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

15177 CX Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>,
 15178 <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, and <wchar.h>.

15179 **APPLICATION USAGE**

15180 None.

15181 **RATIONALE**

15182 None.

15183 **FUTURE DIRECTIONS**

15184 None.

15185 **SEE ALSO**

15186 <ctype.h>, <locale.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>,
15187 <wchar.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *iswalnum()*, *iswalpha()*,
15188 *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
15189 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *towctrans()*, *towlower()*, *towupper()*, *wctrans()*,
15190 *wctype()*

15191 **CHANGE HISTORY**

15192 First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15193 **Issue 6**

15194 The *iswblank()* function is added for alignment with the ISO/IEC 9899:1999 standard.

15195 **NAME**

15196 wordexp.h — word-expansion types

15197 **SYNOPSIS**

15198 #include <wordexp.h>

15199 **DESCRIPTION**15200 The <wordexp.h> header shall define the structures and symbolic constants used by the
15201 wordexp() and wordfree() functions.15202 The structure type **wordexp_t** shall contain at least the following members:

15203 size_t we_wordc Count of words matched by *words*.
 15204 char **we_wordv Pointer to list of expanded words.
 15205 size_t we_offs Slots to reserve at the beginning of *we_wordv*.

15206 The *flags* argument to the wordexp() function shall be the bitwise-inclusive OR of the following
15207 flags:

15208 WRDE_APPEND Append words to those previously generated.
 15209 WRDE_DOOFFS Number of null pointers to prepend to *we_wordv*.
 15210 WRDE_NOCMD Fail if command substitution is requested.
 15211 WRDE_REUSE The *pwordexp* argument was passed to a previous successful call to
 15212 wordexp(), and has not been passed to wordfree(). The result is the same
 15213 as if the application had called wordfree() and then called wordexp()
 15214 without WRDE_REUSE.

15215 WRDE_SHOWERR Do not redirect *stderr* to **/dev/null**.
 15216 WRDE_UNDEF Report error on an attempt to expand an undefined shell variable.

15217 The following constants shall be defined as error return values:

15218 WRDE_BADCHAR One of the unquoted characters—<newline>, ' | ', ' & ', ' ; ', ' < ', ' > ',
 15219 ' (', ') ', ' { ', ' } '—appears in *words* in an inappropriate context.
 15220 WRDE_BADVAL Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.
 15221 WRDE_CMDSUB Command substitution requested when WRDE_NOCMD was set in *flags*.
 15222 WRDE_NOSPACE Attempt to allocate memory failed.

15223 OB XSI WRDE_NOSYS Reserved.

15224 WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated
 15225 string.

15226 The <wordexp.h> header shall define the following type:

15227 XSI **size_t** As described in <stddef.h>.

15228 The following shall be declared as functions and may also be defined as macros. Function
 15229 prototypes shall be provided.

15230 int wordexp(const char *restrict, wordexp_t *restrict, int);
 15231 void wordfree(wordexp_t *);

15232 The implementation may define additional macros or constants using names beginning with
 15233 WRDE_.

15234 **APPLICATION USAGE**

15235 None.

15236 **RATIONALE**

15237 None.

15238 **FUTURE DIRECTIONS**

15239 None.

15240 **SEE ALSO**

15241 <stddef.h>, the System Interfaces volume of IEEE Std 1003.1-2001, *wordexp()*, the Shell and

15242 Utilities volume of IEEE Std 1003.1-2001

15243 **CHANGE HISTORY**

15244 First released in Issue 4. Derived from the ISO POSIX-2 standard.

15245 **Issue 6**

15246 The **restrict** keyword is added to the prototype for *wordexp()*.

15247 The WRDE_NOSYS constant is marked obsolescent.

Index

(time) resolution	79	<poll.h>	287
/	185	<pthread.h>	289
/dev	185	<pwd.h>	294
/dev/console	185	<regex.h>	295
/dev/null	185	<sched.h>	297
/dev/tty	185	<search.h>	299
/tmp	185	<semaphore.h>	301
<aio.h>	206	<setjmp.h>	302
<alert>	36	<signal.h>	303
<arpa/inet.h>	208	<space>	84
<assert.h>	209	<spawn.h>	310
<backspace>	40	<stdarg.h>	312
<blank>	45	<stdbool.h>	314
<carriage-return>	47	<stddef.h>	315
<complex.h>	210	<stdint.h>	316
<cpio.h>	213	<stdio.h>	323
<ctype.h>	214	<stdlib.h>	327
<dirent.h>	216	<string.h>	331
<dlfcn.h>	218	<strings.h>	333
<errno.h>	219	<stropts.h>	334
<fcntl.h>	223	<sys/dir.h>	216
<fenv.h>	226	<sys/ipc.h>	339
<float.h>	229	<sys/mman.h>	341
<fmtmsg.h>	233	<sys/msg.h>	344
<fnmatch.h>	235	<sys/resource.h>	345
<form-feed>	60	<sys/select.h>	347
<ftw.h>	236	<sys/sem.h>	349
<glob.h>	238	<sys/shm.h>	351
<grp.h>	240	<sys/socket.h>	353
<iconv.h>	241	<sys/stat.h>	358
<inttypes.h>	242	<sys/statvfs.h>	362
<iso646.h>	244	<sys/time.h>	364
<langinfo.h>	245	<sys/timeb.h>	366
<libgen.h>	248	<sys/times.h>	367
<limits.h>	249	<sys/types.h>	368
<locale.h>	263	<sys/uio.h>	371
<math.h>	265	<sys/un.h>	372
<monetary.h>	272	<sys/utsname.h>	373
<mqueue.h>	273	<sys/wait.h>	374
<ndbm.h>	275	<syslog.h>	376
<net/if.h>	276	<tab>	89
<netdb.h>	277	<tar.h>	378
<netinet/in.h>	281	<termios.h>	380
<netinet/tcp.h>	285	<tgmath.h>	386
<newline>	68	<time.h>	390
<nl_types.h>	286	<trace.h>	394

<ucontext.h>.....	398	_POSIX minimum values	
<ulimit.h>.....	399	in <limits.h>	255
<unistd.h>	400	_POSIX2_BC_BASE_MAX	253, 257
<utime.h>	419	_POSIX2_BC_DIM_MAX	254, 257
<utmpx.h>	420	_POSIX2_BC_SCALE_MAX	254, 258
<vertical-tab>	95	_POSIX2_BC_STRING_MAX	254, 258
<wchar.h>	422	_POSIX2_CHARCLASS_NAME_MAX.....	254, 258
<wctype.h>	426	_POSIX2_CHAR_TERM	404
<wordexp.h>	428	_POSIX2_COLL_WEIGHTS_MAX	254, 258
±0	97	_POSIX2_C_BIND	404
_Complex_I	210	_POSIX2_C_DEV	404
_CS_PATH	406	_POSIX2_EXPR_NEST_MAX	254, 258
_CS_POSIX_V6_ILP32_OFF32_CFLAGS	406	_POSIX2_FORT_DEV	404
_CS_POSIX_V6_ILP32_OFF32_LDFLAGS	407	_POSIX2_FORT_RUN	404
_CS_POSIX_V6_ILP32_OFF32_LIBS	407	_POSIX2_LINE_MAX	254, 258, 260
_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS	407	_POSIX2_LOCALEDEF	404
_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS	407	_POSIX2_PBS	404
_CS_POSIX_V6_ILP32_OFFBIG_LIBS	407	_POSIX2_PBS_ACCOUNTING	404
_CS_POSIX_V6_LP64_OFF64_CFLAGS	407	_POSIX2_PBS_CHECKPOINT	404
_CS_POSIX_V6_LP64_OFF64_LDFLAGS	407	_POSIX2_PBS_LOCATE	404
_CS_POSIX_V6_LP64_OFF64_LIBS	407	_POSIX2_PBS_MESSAGE	404
_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS	407	_POSIX2_PBS_TRACK	405
_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS	407	_POSIX2_RE_DUP_MAX	251, 254, 258
_CS_POSIX_V6_LPBIG_OFFBIG_LIBS	408	_POSIX2_SW_DEV	405
_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS....	408	_POSIX2_UPE	405
.....	408	_POSIX2_VERSION	400
_CS_XBS5_ILP32_OFF32_CFLAGS	408	_POSIX_ADVISORY_INFO	19, 25-26, 401
_CS_XBS5_ILP32_OFF32_LDFLAGS	408	_POSIX_AIO_LISTIO_MAX	250, 255
_CS_XBS5_ILP32_OFF32_LIBS	408	_POSIX_AIO_MAX	250, 255
_CS_XBS5_ILP32_OFF32_LINTFLAGS	408	_POSIX_ARG_MAX	250, 255
_CS_XBS5_ILP32_OFFBIG_LDFLAGS	408	_POSIX_ASYNCHRONOUS_IO	19, 24-25, 401
_CS_XBS5_ILP32_OFFBIG_LINTFLAGS	408	_POSIX_ASYNC_IO	406
_CS_XBS5_ILP32_OFFBIG_LIBS	408	_POSIX_BARRIERS	19, 27, 401
_CS_XBS5_ILP32_OFFBIG_LINTFLAGS	408	_POSIX_CHILD_MAX	250, 255
_CS_XBS5_LP64_OFF64_CFLAGS	408	_POSIX_CHOWN_RESTRICTED	18, 401
_CS_XBS5_LP64_OFF64_LDFLAGS	408	_POSIX_CLOCKRES_MIN	254
_CS_XBS5_LP64_OFF64_LIBS	408	_POSIX_CLOCK_SELECTION	19, 25-26, 401
_CS_XBS5_LP64_OFF64_LINTFLAGS	408	_POSIX_CPUTIME	19, 25-26, 401
_CS_XBS5_LPBIG_OFFBIG_CFLAGS	408	_POSIX_DELAYTIMER_MAX	250, 255
_CS_XBS5_LPBIG_OFFBIG_LDFLAGS	408	_POSIX_FSYNC	19, 21, 24-25, 401
_CS_XBS5_LPBIG_OFFBIG_LIBS	408	_POSIX_HOST_NAME_MAX	250, 255
_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS	408	_POSIX_IPV6	19
_Imaginary_I	210	_POSIX_JOB_CONTROL	18, 401
_IOFBF	323	_POSIX_LINK_MAX	252, 255
_IOLBF	323	_POSIX_LOGIN_NAME_MAX	250, 255
_IONBF	323	_POSIX_MAPPED_FILES	19, 21, 24, 401
_MIN	249	_POSIX_MAPPED_FILES, ...	25
_PC constants		_POSIX_MAX_CANON	252, 255
defined in <unistd.h>	408	_POSIX_MAX_INPUT	253, 255
_POSIX	249	_POSIX_MEMLOCK	19, 24-25, 401
_POSIX maximum values		_POSIX_MEMLOCK_RANGE	19, 24-25, 401
in <limits.h>	254		

_POSIX_MEMORY_PROTECTION	19, 21, 24-25, 401
_POSIX_MESSAGE_PASSING	19, 24-25, 401
_POSIX_MONOTONIC_CLOCK	19, 25-26, 401
_POSIX_MQ_OPEN_MAX	250, 255
_POSIX_MQ_PRIO_MAX	250, 255
_POSIX_NAME_MAX	253, 256
_POSIX_NGROUPS_MAX	254, 256
_POSIX_NO_TRUNC	18, 102, 402
_POSIX_OPEN_MAX	250, 256
_POSIX_PATH_MAX	253, 256, 372
_POSIX_PIPE_BUF	253, 256
_POSIX_PRIORITIZED_IO	19, 24-25, 402
_POSIX_PRIORITY_SCHEDULING	19, 24-26, 402
_POSIX_PRIO_IO	406
_POSIX_RAW_SOCKETS	19, 402
_POSIX_READER_WRITER_LOCKS	402
_POSIX_REALTIME_SIGNALS	19, 24-25, 402
_POSIX_REGEX	402
_POSIX_RE_DUP_MAX	256
_POSIX_RTSIG_MAX	251, 256
_POSIX_SAVED_IDS	18, 402
_POSIX_SEMAPHORES	19, 24-25, 402
_POSIX_SEM_NSEMS_MAX	251, 256
_POSIX_SEM_VALUE_MAX	251, 256
_POSIX_SHARED_MEMORY_OBJECTS	19, 24-25, 402
_POSIX_SHELL	402
_POSIX_SIGQUEUE_MAX	251, 256
_POSIX_SPAWN	19, 25-26, 402
_POSIX_SPIN_LOCKS	19, 27, 402
_POSIX_SPORADIC_SERVER	19, 25-26, 402
_POSIX_SSIZE_MAX	256, 259
_POSIX_SS_REPL_MAX	251, 256
_POSIX_STREAM_MAX	251, 256
_POSIX_SYMLINK_MAX	253, 256
_POSIX_SYMLOOP_MAX	251, 257
_POSIX_SYNCHRONIZED_IO	19, 24-25, 403
_POSIX_SYNC_IO	406
_POSIX_THREADS	19, 21, 27, 403
_POSIX_THREAD_ATTR_STACKADDR	19, 21, 403
_POSIX_THREAD_ATTR_STACKSIZE	19, 21, 403
_POSIX_THREAD_CPUTIME	19, 27, 403
_POSIX_THREAD_DESTRUCTOR_ITERATIONS	251, 257
_POSIX_THREAD_KEYS_MAX	251, 257
_POSIX_THREAD_PRIORITY_SCHEDULING	19, 26-27, 403
_POSIX_THREAD_PRIO_INHERIT	19, 26, 403
_POSIX_THREAD_PRIO_PROTECT	19, 26, 403
_POSIX_THREAD_PROCESS_SHARED	19, 21, 403
_POSIX_THREAD_SAFE_FUNCTIONS	19, 21, 27, 403
_POSIX_THREAD_SPORADIC_SERVER	19, 27, 403
_POSIX_THREAD_THREADS_MAX	251, 257
_POSIX_TIMEOUTS	20, 25-26, 403
_POSIX_TIMERS	20, 24-27, 403
_POSIX_TIMER_MAX	252, 257
_POSIX_TRACE	20, 27, 403
_POSIX_TRACE_EVENT_FILTER	20, 27, 403
_POSIX_TRACE_EVENT_NAME_MAX	252, 257
_POSIX_TRACE_INHERIT	20, 27, 404
_POSIX_TRACE_LOG	20, 27, 404
_POSIX_TRACE_NAME_MAX	252, 257
_POSIX_TRACE_SYS_MAX	252, 257
_POSIX_TRACE_USER_EVENT_MAX	252, 257
_POSIX_TTY_NAME_MAX	252, 257
_POSIX_TYPED_MEMORY_OBJECTS	20, 25-26, 404
_POSIX_TZNAME_MAX	252, 257
_POSIX_VDISABLE	18, 404
_POSIX_VERSION	18, 400
SC constants	
defined in <unistd.h>	409
_V6_ILP32_OFF32	405
_V6_ILP32_OFFBIG	405
_V6_LP64_OFF64	405
_V6_LPBIG_OFFBIG	405
_XBS5_ILP32_OFF32	405
_XBS5_ILP32_OFFBIG	405
_XBS5_LP64_OFF64	405
_XBS5_LPBIG_OFFBIG	405
_XOPEN_CRYPT	20, 24, 405
_XOPEN_ENH_I18N	405
_XOPEN_IOV_MAX	250, 258
_XOPEN_LEGACY	20, 28, 405
_XOPEN_NAME_MAX	253, 258
_XOPEN_PATH_MAX	253, 258
_XOPEN_REALTIME	20, 24-25, 405
_XOPEN_REALTIME_THREADS	20, 26, 405
_XOPEN_SHM	406
_XOPEN_STREAMS	28, 406
_XOPEN_UNIX	20-21, 406
_XOPEN_VERSION	400
ABDAY	246
ABMON	246
abortive release	35
absolute pathname	35, 102

access mode	35	async-signal-safe function	38
additional file access control mechanism	35	asynchronous events	38
address space	35	asynchronous I/O completion	39
ADV	6	asynchronous I/O operation	39
advanced realtime	25	asynchronous input and output	38
ADVANCED_REALTIME	310	asynchronously-generated signal	39
advanced realtime threads	26	ATEXIT_MAX	250
advisory information	35	attribute selection	383
affirmative response	36	authentication	39
AF_INET	355	authorization	39
AF_INET6	355	background job	39
AF_UNIX	355	background process	39
AF_UNSPEC	355	background process group	39
AIO	6	backquote	40
AIO_ALLDONE	206	BACKREF	179
AIO_CANCELED	206	backslash	40
AIO_LISTIO_MAX	249	backspace character	40
AIO_MAX	250	bandinfo	334
AIO_NOTCANCELED	206	BAR	7
AIO_PRIO_DELTA_MAX	250	barrier	40
AI_ADDRCONFIG	278	base character	40
AI_ALL	278	basename	40
AI_CANONNAME	278	basic regular expression	40, 171
AI_NUMERICHOST	278	batch access list	40
AI_NUMERICSERV	278	batch administrator	41
AI_PASSIVE	278	batch client	41
AI_V4MAPPED	278	batch destination	41
alert	36	batch destination identifier	41
alert character	36	batch directive	41
alias name	36	batch job	41
alignment	36	batch job attribute	42
alternate file access control mechanism	36	batch job identifier	42
alternate signal stack	37	batch job name	42
ALT_DIGITS	246	batch job owner	42
AM_STR	246	batch job priority	42
anchoring	175	batch job state	42
ancillary data	37	batch name service	42
angle brackets	37	batch name space	42
ANYMARK	337	batch node	43
API	37	batch operator	43
application	37	batch queue	43
application address	37	batch queue attribute	43
application conformance	31	batch queue position	43
application program interface	37	batch queue priority	43
appropriate privileges	37	batch rerunability	43
AREGTYPE	378	batch restart	44
argument	38	batch server	44
ARG_MAX	250	batch server name	44
arm (a timer)	38	batch service	44
asterisk	38	batch service request	44
async-cancel-safe function	38	batch submission	44

batch system	44	charmap	
batch target user	45	description	119
batch user	45	CHAR_BIT	258-259
baud rate selection	382	CHAR_MAX	259
BC_BASE_MAX	253	CHAR_MIN	259
BC_DIM_MAX	253	child process	48
BC_SCALE_MAX	254	CHILD_MAX	250
BC_STRING_MAX	254	CHRTYPE	378
BE	7	circumflex	48
bind	45	CLD_CONTINUED	307
blank character	45	CLD_DUMPED	307
blank line	45	CLD_EXITED	307
blkcnt_t	368	CLD_KILLED	307
blksize_t	368	CLD_STOPPED	307
BLKTYPE	378	CLD_TRAPPED	307
block special file	46	CLOCAL	383
block-mode terminal	45	clock	48
blocked process (or thread)	45	clock jump	48
blocking	45	clock tick	48
BOOT_TIME	420	clockid_t	368
braces	46	CLOCKS_PER_SEC	368, 390-391
brackets	46	CLOCK_MONOTONIC	391
BRE (ERE) matching a single character	170	CLOCK_PROCESS_CPUTIME_ID	390
BRE (ERE) matching multiple characters	170	CLOCK_REALTIME	254, 390
BRKINT	381	clock_t	368
broadcast	46	CLOCK_THREAD_CPUTIME_ID	390
BSD	216	CMMSG_DATA	354
BSDLY	382	CMMSG_FIRSTHDR	354
BSn	382	CMMSG_NXTHDR	354
BUFSIZ	323	coded character set	48
built-in	46	codeset	49
built-in utility	46	CODESET	246
BUS_ADRALN	307	collating element	49
BUS_ADRERR	307	collation	49
BUS_OBJERR	307	collation sequence	49
byte	46	COLL_ELEM_MULTI	179
byte input/output functions	47	COLL_ELEM_SINGLE	179
can	5	COLL_WEIGHTS_MAX	254
canonical mode input processing	189	column position	49
carriage-return character	47	COLUMNS	165
CD	7	command	50
character	47	command language interpreter	50
character array	47	complex	210
character class	47	composite graphic symbol	50
character encoding	118	concurrent execution	99
state-dependent	122	condition variable	50
character set	47	conformance	17, 31
character special file	48	POSIX	17
character string	48	POSIX system interfaces	18
CHARCLASS_NAME_MAX	254, 260	XSI	17
		XSI system interfaces	21

conformance document.....	17	C_IXGRP	213
conforming application.....	17	C_IXOTH	213
conforming implementation options	22	C_IXUSR.....	213
connection	50	data segment.....	52
connection mode.....	50	data structure	
connectionless mode.....	50	dirent.....	216
control character.....	51	entry	299
control modes.....	383	group.....	240
control operator	51	lconv.....	263
controlling process	51	msqid_ds	344
controlling terminal	51, 188	stat	358
CONTTYPE	378	tms	367
conversion descriptor	51	utimbuf.....	419
copy	142	data type	
core file.....	51	ACTION	299
CPT	7	cc_t.....	380
CPU	367	DIR.....	216
CPU time	51	div_t	327
clock	52	ENTRY.....	299
timer	52	FILE	324
CRDLY	381	fpos_t	324
CREAD	383	glob_t	238
CRn.....	381	ldiv_t.....	327
CRNCYSTR	246	lldiv_t.....	327
CS	7	mbstate_t.....	422
CSIZE	383	msglen_t.....	344
CSn.....	383	msgqnum_t.....	344
CSTOPB.....	383	nl_catd	286
currency_symbol	142	nl_item.....	286
current job	52	pid_t	303
current working directory	52, 96	ptrdiff_t.....	315
cursor position	52	regex_t	295
CX.....	7	regmatch_t.....	295
C_ constants in <cpio.h>.....	213	regoff_t.....	295
C_IRGRP	213	shmatt_t.....	351
C_IROTH	213	sigset_t.....	303
C_IRUSR.....	213	sig_atomic_t	303
C_ISBLK	213	size_t	315
C_ISCHR.....	213	speed_t.....	380
C_ISCTG.....	213	tcflag_t	380
C_ISDIR.....	213	VISIT	299
C_ISFIFO	213	wchar_t	315
C_ISGID.....	213	wctrans_t.....	426
C_ISLNK	213	wctype_t.....	422
C_ISREG	213	wint_t.....	422
C_ISSOCK.....	213	data types	
C_ISUID.....	213	defined in <fenv.h>.....	226
C_ISVTX.....	213	defined in <sys/types.h>.....	368
C_IWGRP	213	DATEMSK.....	165
C_IWOTH	213	DAY_	246
C_IWUSR	213		

DBL_constants		
defined in <float.h>	230	
DBL_DIG	230, 258	
DBL_EPSILON	231	
DBL_MANT_DIG	230	
DBL_MAX	231, 258	
DBL_MAX_10_EXP	231	
DBL_MAX_EXP	231	
DBL_MIN	232	
DBL_MIN_10_EXP	231	
DBL_MIN_EXP	231	
DBM	275	
DBM_INSERT	275	
DBM_REPLACE	275	
DEAD_PROCESS	420	
DECIMAL_DIG	230	
deferred batch service	52	
DELAYTIMER_MAX	250	
device	52	
output	185	
device ID	52	
dev_t	368	
DIR	216	
directory	53	
directory entry (or link)	53	
directory protection	99	
directory stream	53	
dirent structure	216	
DIRTYPE	378	
disarm (a timer)	53	
display	53	
display line	53	
documentation	17	
dollar sign	53	
domain error	107	
dot	53	
dot-dot	54	
double-quote	54	
downshifting	54	
driver	54	
DUP_COUNT	179	
D_FMT	246	
D_T_FMT	246	
E2BIG	219	
EACCES	219	
EADDRINUSE	219	
EADDRNOTAVAIL	219	
EAFNOSUPPORT	219	
EAGAIN	219	
EAI_AGAIN	279	
EAI_BADFLAGS	279	
EAI_FAIL	279	
EAI_FAMILY	279	
EAI_MEMORY	279	
EAI_NONAME	279	
EAI_OVERFLOW	279	
EAI_SERVICE	279	
EAI_SOCKTYPE	279	
EAI_SYSTEM	279	
EALREADY	219	
EBADF	219	
EBADMSG	219	
EBUSY	219	
ECANCELED	219	
ECHILD	219	
ECHO	383	
ECHOE	383	
ECHOK	383	
ECHONL	383	
ECONNABORTED	219	
ECONNREFUSED	219	
ECONNRESET	219	
EDEADLK	219	
EDESTADDRREQ	219	
EDOM	219	
EDQUOT	219	
EEXIST	219	
EFAULT	219	
EFBIG	219	
effective group ID	54	
effective user ID	54	
EHOSTUNREACH	219	
EIDRM	219	
eight-bit transparency	54	
EILSEQ	219	
EINPROGRESS	220	
EINTR	220	
EINVAL	220	
EIO	220	
EISCONN	220	
EISDIR	220	
ELOOP	220	
EMFILE	220	
EMLINK	220	
EMPTY	420	
empty directory	54	
empty line	55	
empty string (or null string)	55	
empty wide-character string	55	
EMSGSIZE	220	
EMULTIHOP	220	
ENAMETOOLONG	220	

encoding		
character.....	118	
encoding rule.....	55	
encryption.....	24	
ENETDOWN.....	220	
ENETRESET.....	220	
ENETUNREACH.....	220	
ENFILE.....	220	
ENOBUS.....	220	
ENODATA.....	220	
ENODEV.....	220	
ENOENT.....	220	
ENOEXEC.....	220	
ENOLCK.....	220	
ENOLINK.....	220	
ENOMEM.....	220	
ENOMSG.....	220	
ENOPROTOOPT.....	220	
ENOSPC.....	220	
ENOSR.....	220	
ENOSTR.....	220	
ENOSYS.....	220	
ENOTCONN.....	220	
ENOTDIR.....	220	
ENOTEMPTY.....	220	
ENOTSOCK.....	220	
ENOTSUP.....	221	
ENOTTY.....	221	
entire regular expression.....	55, 169	
environment variables		
internationalization.....	162	
ENXIO.....	221	
EOF.....	323	
EOPNOTSUPP.....	221	
EOVERFLOW.....	221	
EPERM.....	221	
EPIPE.....	221	
epoch.....	55	
EPROTO.....	221	
EPROTONOSUPPORT.....	221	
EPROTOTYPE.....	221	
equivalence class.....	55	
era.....	55	
ERA.....	246	
ERANGE.....	221	
ERA_D_FMT.....	246	
ERA_D_T_FMT.....	246	
ERA_T_FMT.....	246	
EROFS.....	221	
error conditions		
mathematical functions.....	107	
ESPIPE.....	221	
ESRCH.....	221	
ESTALE.....	221	
ETIME.....	221	
ETIMEDOUT.....	221	
ETXTBSY.....	221	
event management.....	56	
EWouldBlock.....	221	
EXDEV.....	221	
executable file.....	56	
execute.....	56	
execution time.....	51, 56	
measurement.....	101	
monitoring.....	56	
EXIT_FAILURE.....	327	
EXIT_SUCCESS.....	327	
expand.....	56	
EXPR_NEST_MAX.....	254	
extended regular expression.....	56, 175	
extended security controls.....	57, 99	
extension		
CX.....	7	
OH.....	10	
XSI.....	14	
F-LOCK.....	408	
FD.....	7	
FD_CLOEXEC.....	223	
FD_CLR.....	347	
FD_ISSET.....	347	
FD_SET.....	347	
fd_set.....	347 , 364	
FD_SETSIZE.....	347	
FD_ZERO.....	347	
feature test macro.....	57	
fenv_t.....	226	
fexcept_t.....	226	
FE_constants		
defined in <fenv.h>.....	226	
FE_ALL_EXCEPT.....	226	
FE_DFL_ENV.....	227	
FE_DIVBYZERO.....	226	
FE_DOWNWARD.....	226	
FE_INEXACT.....	226	
FE_INVALID.....	226	
FE_OVERFLOW.....	226	
FE_TONEAREST.....	226	
FE_TOWARDZERO.....	226	
FE_UNDERFLOW.....	226	
FE_UPWARD.....	226	
FFDLY.....	382	
FFn.....	382	

field	57	FNM_constants	
FIFO	57	in <fnmatch.h>	235
FIFO special file	57	FNM_NOESCAPE	235
FIFOTYPE	378	FNM_NOMATCH	235
file	57	FNM_NOSYS	235
FILE	323, 422	FNM_PATHNAME	235
file access permissions	99	FNM_PERIOD	235
file characteristics		FOPEN_MAX	251, 323
data structure	360	foreground job	60
header	360	foreground process	60
file description	58	foreground process group	60
file descriptor	58	foreground process group ID	60
file group class	58	form-feed character	60
file hierarchy	100	format of entries	205
file mode	58	FPE_FLTDIV	307
file mode bits	58	FPE_FLTINV	307
file offset	59	FPE_FLTOVF	307
file other class	59	FPE_FLTRES	307
file owner class	59	FPE_FLTSUB	307
file permission bits	59	FPE_FLTUND	307
file serial number	59	FPE_INTDIV	307
file system	59	FPE_INTOVF	307
file times update	100	FR	7
file type	59	frac_digits	143
filename	58	fsblkcnt_t	368
filename portability	58	FSC	8
FILENAME_MAX	323	fsfilcnt_t	368
FILESIZEBITS	252	FTW	236
filter	60	FTW_constants	
FIPS	18	in <ftw.h>	236
first open (of a file)	60	FTW_CHDIR	236
flow control	60	FTW_D	236
FLT_constants		FTW_DEPTH	236
defined in <float.h>	230	FTW_DNR	236
FLT_DIG	230, 258	FTW_DP	236
FLT_EPSILON	231	FTW_F	236
FLT_EVAL_METHOD	229	FTW_MOUNT	236
FLT_MANT_DIG	230	FTW_NS	236
FLT_MAX	231, 258	FTW_PHYS	236
FLT_MAX_10_EXP	231	FTW_SL	236
FLT_MAX_EXP	231	FTW_SLN	236
FLT_MIN	232	F_DUPFD	223
FLT_MIN_10_EXP	231	F_GETFD	223
FLT_MIN_EXP	231	F_GETFL	223
FLT_RADIX	230	F_GETLK	223
FLT_ROUNDS	229	F_GETOWN	223
FLUSHR	336	F_OK	406
FLUSHRW	336	F_RDLCK	223
FLUSHW	336	F_SETFD	223
FMNAMESZ	335-336	F_SETFL	223
		F_SETLK	223

F_SETLKW.....	223	IGNCR.....	381
F_SETOWN.....	223	IGNPAR.....	381
F_TEST.....	408	ILL_BADSTK.....	307
F_TLOCK.....	408	ILL_COPROC.....	307
F_ULOCK.....	408	ILL_ILADR.....	307
F_UNLCK.....	223	ILL_ILLOPC.....	307
F_WRLCK.....	223	ILL_ILLOPN.....	307
GETALL.....	349	ILL_ILLTRP.....	307
GETNCNT.....	349	ILL_PRVOPC.....	307
GETPID.....	349	ILL_PRVREG.....	307
GETVAL.....	349	imaginary.....	210
GETZCNT.....	349	implementation-defined.....	5
gid_t.....	368	IN6_IS_ADDR_LINKLOCAL.....	283
GLOB_constants		IN6_IS_ADDR_LOOPBACK.....	283
defined in <glob.h>.....	238	IN6_IS_ADDR_MC_GLOBAL.....	283
GLOB_ABORTED.....	238	IN6_IS_ADDR_MC_LINKLOCAL.....	283
GLOB_APPEND.....	238	IN6_IS_ADDR_MC_NODELOCAL.....	283
GLOB_DOOFFS.....	238	IN6_IS_ADDR_MC_ORGLOCAL.....	283
GLOB_ERR.....	238	IN6_IS_ADDR_MC_SITELOCAL.....	283
GLOB_MARK.....	238	IN6_IS_ADDR_MULTICAST.....	283
GLOB_NOCHECK.....	238	IN6_IS_ADDR_SITELOCAL.....	283
GLOB_NOESCAPE.....	238	IN6_IS_ADDR_UNSPECIFIED.....	283
GLOB_NOMATCH.....	238	IN6_IS_ADDR_V4COMPAT.....	283
GLOB_NOSORT.....	238	IN6_IS_ADDR_V4MAPPED.....	283
GLOB_NOSPACE.....	238	INADDR_ANY.....	282
GLOB_NOSYS.....	238	INADDR_BROADCAST.....	282
grammar		incomplete line.....	62
locale.....	153	INET6_ADDRSTRLEN.....	282
regular expression.....	179	INET_ADDRSTRLEN.....	282
graphic character.....	61	Inf.....	62
group database.....	61	INFINITY.....	266
group ID.....	61	INIT_PROCESS.....	420
group name.....	61	INLCR.....	381
hard limit.....	61	ino_t.....	368
hard link.....	61	INPCK.....	381
headers.....	205	instrumented application.....	62
HOME.....	165	interactive shell.....	62
home directory.....	61	internationalization.....	62
host byte order.....	62, 101	interprocess communication.....	62
HOST_NAME_MAX.....	250	INTMAX_MAX.....	320
HUGE_VAL.....	266	INTMAX_MIN.....	320
HUGE_VALF.....	266	INTN_MAX.....	319
HUGE_VALL.....	266	INTN_MIN.....	319
HUPCL.....	383	INTPTR_MAX.....	319
I.....	210	INTPTR_MIN.....	319
ICANON.....	383	int_curr_symbol.....	142
ICRNL.....	381	INT_FASTN_MAX.....	319
idtype_t.....	374	INT_FASTN_MIN.....	319
id_t.....	368	int_frac_digits.....	143
IEXTEN.....	383	INT_LEASTN_MAX.....	319
IGNBRK.....	381	INT_LEASTN_MIN.....	319

INT_MAX.....	259	I_FIND.....	335
INT_MIN.....	260	I_FLUSH.....	335
int_n_cs_precedes.....	144	I_FLUSHBAND.....	335
int_n_sep_by_space.....	144	I_GETBAND.....	335
int_n_sign_posn.....	144	I_GETCLTIME.....	335
int_p_cs_precedes.....	144	I_GETSIG.....	335
int_p_sep_by_space.....	144	I_GRDOPT.....	335
int_p_sign_posn.....	144	I_GWROPT.....	335
invalid.....	170	I_LINK.....	335
invariant values.....	260	I_LIST.....	335
invoke.....	62	I_LOOK.....	335
iovec.....	371	I_NREAD.....	335
IOV_MAX.....	250, 371	I_PEEK.....	335
IP6.....	8	I_PLINK.....	335
IPC.....	339	I_POP.....	335
IPC_constants		I_PUNLINK.....	335
defined in <sys/ipc.h>.....	339	I_PUSH.....	335
IPC_CREAT.....	339	I_RECVFD.....	335
IPC_EXCL.....	339	I_SENDFD.....	335
IPC_NOWAIT.....	339	I_SETCLTIME.....	335
IPC_PRIVATE.....	339	I_SETSIG.....	335
IPC_RMID.....	339	I_SRDOPT.....	335
IPC_SET.....	339	I_STR.....	335
IPC_STAT.....	339	I_SWROPT.....	335
IPPROTO_ICMP.....	282	I_UNLINK.....	335
IPPROTO_IP.....	282	job.....	63
IPPROTO_IPV6.....	282	job control.....	63
IPPROTO_RAW.....	282	job control job ID.....	63
IPPROTO_TCP.....	282	key_t.....	368
IPPROTO_UDP.....	282	LANG.....	162
IPV6_JOIN_GROUP.....	282	last close (of a file).....	63
IPV6_LEAVE_GROUP.....	283	LASTMARK.....	337
IPV6_MULTICAST_HOPS.....	283	LC_ALL.....	163 , 263
IPV6_MULTICAST_IF.....	283	LC_COLLATE.....	163 , 254, 263
IPV6_MULTICAST_LOOP.....	283	description.....	134
IPV6_UNICAST_HOPS.....	283	LC_CTYPE.....	163 , 246, 263, 426
IPV6_V6ONLY.....	283	description.....	126
ISIG.....	383	LC_MESSAGES.....	163 , 246, 263, 286
ISO C standard.....	210	description.....	152
ISTRIP.....	381	LC_MONETARY.....	163 , 246, 263
itimerval.....	364	description.....	142
ITIMER_PROF.....	364	LC_NUMERIC.....	163 , 246, 263
ITIMER_REAL.....	364	description.....	145
ITIMER_VIRTUAL.....	364	LC_TIME.....	163 , 246, 263
IXANY.....	381	description.....	147
IXOFF.....	381	LDBL_constants	
IXON.....	381	defined in <float.h>.....	230
I_ATMARK.....	335	LDBL_DIG.....	231
I_CANPUT.....	335	LDBL_EPSILON.....	232
I_CKBAND.....	335	LDBL_MANT_DIG.....	230
I_FDINSERT.....	335	LDBL_MAX.....	231

LDBL_MAX_10_EXP	231	LOG_MASK.....	376
LDBL_MAX_EXP.....	231	LOG_NDELAY.....	376
LDBL_MIN.....	232	LOG_NEWS.....	376
LDBL_MIN_10_EXP	231	LOG_NOTICE.....	377
LDBL_MIN_EXP.....	231	LOG_NOWAIT	376
legacy	5, 28	LOG_ODELAY	376
limit		LOG_PID.....	376
numerical	258	LOG_USER	376
line	63	LOG_UUCP	376
line control	384	LOG_WARNING	377
LINES	165	LONG_BIT	258-259
LINE_MAX	254	LONG_MAX.....	259
linger.....	63	LONG_MIN.....	260
link	63	lower multiplexing.....	85
link count.....	64	L_ANCHOR	179
LINK_MAX.....	252	L_ctermid	323
LIO_NOP.....	206	L_tmpnam.....	323
LIO_NOWAIT	206	MAGIC	213
LIO_READ	206	map	64
LIO_WAIT.....	206	MAP_FIXED	341
LIO_WRITE	206	MAP_PRIVATE	341
LLONG_MAX	260	MAP_SHARED	341
LLONG_MIN	260	margin codes	
LNKTYPE.....	378	notation.....	14
local customs	64	marked message	65
local IPC.....	64	matched.....	65, 169
local modes	383	mathematical functions	
locale	64, 123	domain error.....	107
grammar.....	153	error conditions	107
POSIX.....	124	NaN arguments	108
locale definition.....	124	pole error.....	108
localization.....	64	range error	108
login.....	64	MAXARGS.....	313
login name.....	64	MAXFLOAT.....	266
LOGIN_NAME_MAX.....	250	maximum values	254
LOGIN_PROCESS.....	420	MAX_CANON.....	252
LOGNAME.....	166	MAX_INPUT	252
LOG_ALERT	377	may	5
LOG_AUTH.....	376	MB_CUR_MAX.....	327
LOG_CONS.....	376	MB_LEN_MAX	258-259
LOG_CRIT	377	MC1	8
LOG_CRON	376	MC2	8
LOG_DAEMON	376	MC3	8
LOG_DEBUG	377	MCL_CURRENT	341
LOG_EMERG	377	MCL_FUTURE.....	341
LOG_ERR.....	377	mcontext_t	398
LOG_INFO.....	377	memory mapped files.....	65
LOG_KERN	376	memory object.....	65
LOG_LOCAL.....	376	memory synchronization.....	102
LOG_LPR.....	376	memory-resident	65
LOG_MAIL	376	message.....	65

message catalog	66	MSG_ANY	337
message catalog descriptor	66	MSG_BAND	337
message queue	66	MSG_CTRUNC	355
META_CHAR	179	MSG_DONTROUTE	355
MF	8	MSG_EOR	355
minimum values	255	MSG_HIPRI	337
MINSIGSTKSZ	305	MSG_NOERROR	344
ML	9	MSG_OOB	355
MLR	9	MSG_PEEK	355
MM_macros	233	MSG_TRUNC	355
MM_APPL	233	MSG_WAITALL	355
MM_CONSOLE	233	MS_ASYNC	341
MM_ERROR	233	MS_INVALIDATE	341
MM_FIRM	233	MS_SYNC	341
MM_HALT	233	multi-character collating element	66
MM_HARD	233	mutex	66
MM_INFO	233	MUXID_ALL	337
MM_NOCON	234	MX	9
MM_NOMSG	233	M_	265
MM_NOSEV	233	M_E	265
MM_NOTOK	233	M_LN	265
MM_NRECOV	233	M_LOG10E	265
MM_NULLACT	233	M_LOG2E	265
MM_NULLLBL	233	M_PI	265
MM_NULLMC	233	M_SQRT1_2	266
MM_NULLSEV	233	M_SQRT2	266
MM_NULLTAG	233	name	67
MM_NULLTXT	233	named STREAM	67
MM_OK	233	NAME_MAX	102, 216, 253
MM_OPSYS	233	NaN (Not a Number)	67, 229, 266
MM_PRINT	233	NaN arguments	
MM_RECOVER	233	mathematical functions	108
MM_SOFT	233	native language	67
MM_UTIL	233	NCCS	380
MM_WARNING	233	NDEBUG	209
mode	66	negative response	67
mode_t	368	negative_sign	143
MON	9	network	67
monotonic clock	66	network address	67
MON_	246	network byte order	68, 101
mon_decimal_point	142	newline character	68
mon_grouping	143	NEW_TIME	420
mon_thousands_sep	143	NGROUPS_MAX	254
MORECTL	337	nice value	68
MOREDATA	337	NI_DGRAM	279
mount point	66	NI_NAMEREQD	279
MPR	9	NI_NOFQDN	278
MQ_OPEN_MAX	250	NI_NUMERICHOST	278
MQ_PRIO_MAX	250	NI_NUMERICSERV	279
MSG	9	NLDLY	381
MSGVERB	166	nlink_t	368

NLn.....	381	BAR.....	7
NLSPATH.....	163	BE.....	7
NL_ARGMAX.....	260	CD.....	7
NL_CAT_LOCALE.....	286	CPT.....	7
NL_LANGMAX.....	260	CS.....	7
NL_MSGMAX.....	260	FD.....	7
NL_NMAX.....	260	FR.....	7
NL_SETD.....	286	FSC.....	8
NL_SETMAX.....	260	IP6.....	8
NL_TEXTMAX.....	260	MC1.....	8
NOEXPR.....	246	MC2.....	8
NOFLSH.....	383	MC3.....	8
non-blocking.....	68	MF.....	8
non-canonical mode input processing.....	190	ML.....	9
non-spacing characters.....	68	MLR.....	9
NOSTR.....	246	MON.....	9
NUL.....	68	MPR.....	9
NULL.....	315, 323, 327, 331, 390, 406	MSG.....	9
null byte.....	69	MX.....	9
null pointer.....	69	PIO.....	10
null string.....	69	PS.....	10
null wide-character code.....	69	RS.....	10
number sign.....	69	RTS.....	10
numerical limits.....	258	SD.....	10
NZERO.....	261	SEM.....	10
n_cs_precedes.....	143	SHM.....	11
n_sep_by_space.....	143	SIO.....	11
n_sign_posn.....	144	SPI.....	11
OB.....	9	SPN.....	11
object file.....	69	SS.....	11
OCRNL.....	381	TCT.....	11
octet.....	69	TEF.....	11
OF.....	10	THR.....	12
offset maximum.....	69	TMO.....	12
off_t.....	368	TMR.....	12
OFILL.....	381	TPI.....	12
OH.....	10	TPP.....	12
OLD_TIME.....	420	TPS.....	12
ONLCR.....	381	TRC.....	12
ONLRET.....	381	TRI.....	12
ONOCR.....	381	TRL.....	13
opaque address.....	69	TSA.....	13
open file.....	70	TSF.....	13
open file description.....	70	TSH.....	13
OPEN_MAX.....	250, 301	TSP.....	13
operand.....	70	TSS.....	13
operator.....	70	TYM.....	13
OPOST.....	381	UP.....	14
option.....	70	XSR.....	14
ADV.....	6	option-argument.....	70
AIO.....	6		

options		
shell and utilities.....	29	
system interfaces	29	
ORD_CHAR	179	
orientation	70	
orphaned process group	70	
output devices.....	185	
O_ constants		
defined in <fcntl.h>	223-224	
O_ACCMODE.....	224	
O_APPEND	223	
O_CREAT	223	
O_DSYNC	223	
O_EXCL.....	223	
O_NOCTTY	223	
O_NONBLOCK	223	
O_RDONLY	224	
O_RDWR.....	224	
O_RSYNC	224	
O_SYNC	224	
O_TRUNC.....	223	
O_WRONLY	224	
page	71	
page size	71	
PAGESIZE.....	250	
PAGE_SIZE.....	251	
parameter	71	
PARENB	383	
parent directory	71	
parent process	71	
parent process ID.....	71	
PARMRK	381	
PARODD	383	
PATH.....	166	
path prefix	72	
pathname.....	72	
pathname component.....	72	
pathname resolution.....	102	
pathname variable values.....	252	
PATH_MAX.....	253, 261	
pattern.....	72	
period	72	
permissions	72	
persistence.....	72	
pid_t	368	
PIO	10	
pipe.....	73	
PIPE_BUF	253	
PM_STR.....	246	
pole error	108	
POLLERR	287	
pollfd	287	
POLLHUP	287	
POLLIN	287	
polling	73	
POLLNVAL	287	
POLLOUT	287	
POLLPRI.....	287	
POLLRDBAND	287	
POLLRDNORM.....	287	
POLLWRBAND	287	
POLLWRNORM.....	287	
POLL_ERR	307	
POLL_HUP	307	
POLL_IN	307	
POLL_MSG	307	
POLL_OUT	307	
POLL_PRI.....	307	
portable character set	73, 115	
portable filename character set	73, 100	
positional parameter.....	73	
positive_sign.....	143	
POSIX		
conformance.....	17	
POSIX locale	124	
POSIX shell and utilities.....	20	
POSIX system interfaces		
conformance.....	18	
POSIX2_CHAR_TERM	20, 29	
POSIX2_C_DEV.....	20, 29	
POSIX2_FORT_DEV.....	20, 29	
POSIX2_FORT_RUN	20, 29	
POSIX2_LOCALEDEF.....	20, 30	
POSIX2_PBS	20, 30	
POSIX2_PBS_ACCOUNTING	20, 30	
POSIX2_PBS_CHECKPOINT	30	
POSIX2_PBS_LOCATE.....	20, 30	
POSIX2_PBS_MESSAGE.....	20, 30	
POSIX2_PBS_TRACK.....	20, 30	
POSIX2_SW_DEV	20, 30	
POSIX2_UPE	20, 30	
POSIX_ALLOC_SIZE_MIN	253	
POSIX_FADV_DONTNEED	224	
POSIX_FADV_NOREUSE	224	
POSIX_FADV_NORMAL	224	
POSIX_FADV_RANDOM	224	
POSIX_FADV_SEQUENTIAL	224	
POSIX_FADV_WILLNEED.....	224	
POSIX_MADV_DONTNEED	342	
POSIX_MADV_NORMAL.....	341	
POSIX_MADV_RANDOM.....	342	
POSIX_MADV_SEQUENTIAL.....	342	

POSIX_MADV_WILLNEED	342	PTHREAD_CANCEL_DEFERRED	289
POSIX_REC_INCR_XFER_SIZE	253	PTHREAD_CANCEL_DISABLE	289
POSIX_REC_MAX_XFER_SIZE	253	PTHREAD_CANCEL_ENABLE	289
POSIX_REC_MIN_XFER_SIZE	253	PTHREAD_COND_INITIALIZER	289
POSIX_TYPED_MEM_ALLOCATE	342	PTHREAD_CREATE_DETACHED	289
POSIX_TYPED_MEM_ALLOCATE_CONTIG.....	342	PTHREAD_CREATE_JOINABLE	289
POSIX_TYPED_MEM_MAP_ALLOCATABLE	342	PTHREAD_DESTRUCTOR_ITERATIONS	251
preallocation	73	PTHREAD_EXPLICIT_SCHED.....	289
preempted process (or thread)	74	PTHREAD_INHERIT_SCHED.....	289
previous job	74	PTHREAD_KEYS_MAX	251
printable character.....	74	PTHREAD_MUTEX_DEFAULT.....	289
printable file.....	74	PTHREAD_MUTEX_ERRORCHECK	289
priority	74	PTHREAD_MUTEX_INITIALIZER	289
priority band.....	74	PTHREAD_MUTEX_NORMAL	289
priority inversion	74	PTHREAD_MUTEX_RECURSIVE	289
priority scheduling.....	74	PTHREAD_ONCE_INIT	289
priority-based scheduling.....	75	PTHREAD_PRIO_INHERIT	289
PRIO_ constants		PTHREAD_PRIO_NONE.....	289
defined in <sys/resource.h>	345	PTHREAD_PRIO_PROTECT	289
PRIО_PGRP	345	PTHREAD_PROCESS_PRIVATE.....	289
PRIО_PROCESS.....	345	PTHREAD_PROCESS_SHARED.....	289
PRIО_USER	345	PTHREAD_RWLOCK_INITIALIZER	289
privilege.....	75	PTHREAD_SCOPE_PROCESS.....	289
process	75	PTHREAD_SCOPE_SYSTEM	289
process group	75	PTHREAD_STACK_MIN	251
process group ID.....	75	PTHREAD_THREADS_MAX.....	251
process group leader.....	75	PTRDIFF_MAX	320
process group lifetime.....	75	PTRDIFF_MIN	320
process ID.....	76	PWD	166
process ID reuse.....	103	P_ALL	374
process lifetime	76	p_cs_precedes	143
process memory locking	76	P_GID	374
process termination.....	76	P_PID	374
process virtual time.....	76	p_sep_by_space	143
process-to-process communication	76	p_sign_posn.....	143
program	77	P_tmpdir.....	323
protocol.....	77	quiet NaN.....	229
PROT_EXEC	341	QUOTED_CHAR	179
PROT_NONE.....	341	radix character.....	77
PROT_READ	341	RADIXCHAR	246
PROT_READ constants		RAND_MAX	327
in <sys/mman.h>	341	range error.....	108
PROT_WRITE	341	result overflows	108
PS.....	10	result underflows	108
pseudo-terminal.....	77	read-only file system.....	77
PTHREAD_BARRIER_SERIAL_THREAD	289	read-write lock	77
PTHREAD_CANCELED	289	real group ID.....	77
PTHREAD_CANCEL_ASYNCHRONOUS	289	real user ID.....	78
		realtime.....	24, 78
		REALTIME	206, 273, 297, 301
		realtime signal extension	78

REALTIME THREADS	26	RLIMIT_NOFILE	346
realtime threads	26	RLIMIT_STACK	346
record	78	RLIM_INFINITY	345
redirection	78	RLIM_SAVED_CUR	345
redirection operator	78	RLIM_SAVED_MAX	345
reentrant function	78	RMSGD	336
referenced shared memory object	78	RMSGN	336
refresh	79	RNORM	336
region	79	root directory	80
REGTYPE	378	RPROTDAT	336
regular expression	79	RPROTDIS	336
basic	171	RPROTNORM	336
extended	175	RS	10
grammar	179	RS_HIPRI	336
regular file	79	RTLD_GLOBAL	218
REG_ constants		RTLD_LAZY	218
defined in <regex.h>	295	RTLD_LOCAL	218
REG_BADBR	295	RTLD_NOW	218
REG_BADPAT	295	RTS	10
REG_BADRPT	296	RTSIG_MAX	251, 304
REG_EBRACE	295	runnable process (or thread)	80
REG_EBRACK	295	running process (or thread)	80
REG_ECOLLATE	295	runtime values	
REG_ECTYPE	295	increasable	253
REG_EESCAPE	295	invariant	249
REG_ENOSYS	296	rusage	345
REG_EPAREN	295	RUSAGE_CHILDREN	345
REG_ERANGE	296	RUSAGE_SELF	345
REG_ESPACE	296	R_ANCHOR	179
REG_ESUBREG	295	R_OK	406
REG_EXTENDED	295	saved resource limits	80
REG_ICASE	295	saved set-group-ID	80
REG_NEWLINE	295	saved set-user-ID	80
REG_NOMATCH	295	SA_ constants	
REG_NOSUB	295	declared in <signal.h>	305
REG_NOTBOL	295	SA_NOCLDSTOP	305
REG_NOTEOL	295	SA_NOCLDWAIT	305
relative pathname	79, 102	SA_NODEFER	305
relocatable file	79	SA_ONSTACK	305
relocation	79	SA_RESETHAND	305
requested batch service	79	SA_RESTART	305
requirements	17	SA_SIGINFO	305
result overflows	108	SCHAR_MAX	259
result underflows	108	SCHAR_MIN	259-260
RE_DUP_MAX	251, 254	scheduling	80
rlimit	345	scheduling allocation domain	80
RLIMIT_AS	346	scheduling contention scope	81
RLIMIT_CORE	345	scheduling policy	81, 103
RLIMIT_CPU	345	SCHED_FIFO	297
RLIMIT_DATA	345	SCHED_OTHER	297
RLIMIT_FSIZE	345	SCHED_RR	297

SCHED_SPORADIC	297	SIGINT	304
SCM_RIGHTS	354	SIGKILL	304
screen	81	signal	82
scroll	81	signal stack	83
SD	10	signaling NaN	229
seconds since the Epoch	104	SIGPIPE	304
SEEK_CUR	223, 323, 408	SIGPOLL	304, 307
SEEK_END	223, 323, 408	SIGPROF	304
SEEK_SET	223, 323, 408	SIGQUEUE_MAX	251
SEGV_ACCERR	307	SIGQUIT	304
SEGV_MAPERR	307	SIGRTMAX	303
SEM	10	SIGRTMIN	303
semaphore	81, 104	SIGSEGV	304, 307
semaphore lock operation	104	SIGSTKSZ	305
semaphore unlock operation	105	SIGSTOP	304
SEM_NSEMS_MAX	251	SIGSYS	304
SEM_UNDO	349	SIGTERM	304
SEM_VALUE_MAX	251	SIGTRAP	304, 307
session	81	SIGTSTP	304
session leader	82	SIGTTIN	304
session lifetime	82	SIGTTOU	304
SETALL	349	SIGURG	304
SETVAL	349	SIGUSR1	304
shall	5	SIGUSR2	304
shared memory object	82	SIGVTALRM	304
shell	82	SIGXCPU	304
SHELL	166	SIGXFSZ	304
shell script	82	SIG_ATOMIC_MAX	320
shell, the	82	SIG_ATOMIC_MIN	320
SHM	11	SIG_BLOCK	305
SHMLBA	351	SIG_DFL	303
SHM_RDONLY	351	SIG_ERR	303
SHM_RND	351	SIG_HOLD	303
should	5	SIG_IGN	303
SHRT_MAX	259	SIG_SETMASK	305
SHRT_MIN	260	SIG_UNBLOCK	305
SHUT_RD	355	single-quote	83
SHUT_RDWR	356	SIO	11
SHUT_WR	356	SIZE_MAX	320
SIGABRT	304	size_t	331, 368
SIGALRM	304	SI_ASYNCIO	307
SIGBUS	304, 307	SI_MESGQ	307
SIGCHLD	304, 307	SI_QUEUE	307
SIGCONT	304	SI_TIMER	307
SIGEV_NONE	303	SI_USER	307
SIGEV_SIGNAL	303	slash	83
SIGEV_THREAD	303	SNDZERO	337
SIGFPE	304, 307	socket	83
SIGHUP	304	socket address	83
SIGILL	304, 307	SOCK_DGRAM	354
siginfo_t	306	SOCK_RAW	354

SOCK_SEQPACKET	354	stream	85
SOCK_STREAM	354	STREAM	220
soft limit	83	STREAM end	85
SOL_SOCKET	354	STREAM head	85
SOMAXCONN	355	STREAMS	28, 334
source code	83	STREAMS multiplexor	85
SO_ACCEPTCONN	354	STREAM_MAX	251
SO_BROADCAST	355	strfdinsert	334
SO_DEBUG	355	string	85
SO_DONTROUTE	355	striocctl	334
SO_ERROR	355	strpeek	334
SO_KEEPALIVE	355	strrecvfd	334
SO_LINGER	355	str_list	334
SO_OOBINLINE	355	str_mlist	335
SO_RCVBUF	355	ST_NOSUID	362
SO_RCVLOWAT	355	ST_RDONLY	362
SO_RCVTIMEO	355	subprofiling	22
SO_REUSEADDR	355	subshell	86
SO_SNDBUF	355	successfully transferred	86
SO_SNDLOWAT	355	supplementary group ID	86
SO_SNDTIMEO	355	suseconds_t	368
SO_TYPE	355	suspended job	86
space character	84	symbolic link	86
spawn	84	SYMLINK_MAX	253, 261
special built-in	84	SYMLOOP_MAX	251
special parameter	84	SYMTYPE	378
SPEC_CHAR	180	synchronized I/O completion	86
SPI	11	synchronized I/O data integrity completion	87
spin lock	84	synchronized I/O file integrity completion	87
SPN	11	synchronized I/O operation	87
sporadic server	84	synchronized input and output	86
SS	11	synchronous I/O operation	87
SSIZE_MAX	259, 369	synchronously-generated signal	87
ssize_t	368	system	87
SS_DISABLE	305	system console	88
SS_ONSTACK	305	system crash	88
SS_REPL_MAX	251	system databases	88
stack_t	306	system documentation	88
standard error	84	system process	88
standard input	84	system reboot	89
standard output	84	system trace event	89
standard utilities	85	system-wide	89
stat data structure	358	S_ constants	
stderr	323	defined in <sys/stat.h>	358-359
STDERR_FILENO	411	S_ macros	
stdin	323	defined in <sys/stat.h>	359
STDIN_FILENO	411	S_BANDURG	336
stdout	324	S_ERROR	336
STDOUT_FILENO	411	S_HANGUP	336
strbuf	334	S_HIPRI	336
STREAM	85	S_IFBLK	358

S_IFCHR.....	359	TCSAFLUSH	383
S_IFDIR.....	359	TCSANOW	383
S_IFIFO	359	TCT	11
S_IFLNK	359	TEF	11
S_IFMT.....	358	TERM	166
S_IFREG.....	359	terminal	
S_IFSOCK.....	359	controlling.....	188
S_INPUT.....	336	terminal (or terminal device)	89
S_IRGRP	359	terminal types.....	185
S_IROTH	359	termios	187
S_IRUSR	359	canonical mode input processing	189
S_IRWXG	359	control modes.....	196
S_IRWXO	359	controlling terminal	188
S_IRWXU	359	input modes.....	193
S_ISBLK	359	local modes.....	197
S_ISCHR.....	359	non-canonical mode input processing.....	190
S_ISDIR.....	360	output modes.....	194
S_ISFIFO	360	special control characters	198
S_ISGID.....	359-360	text column	89
S_ISLNK	360	text file.....	90
S_ISREG.....	360	TGEXEC.....	378
S_ISSOCK.....	360	TGREAD.....	378
S_ISUID.....	359-360	TGWRITE.....	378
S_ISVTX.....	359	THOUSEP	246
S_IWGRP.....	359	THR	12
S_IWOTH.....	359	thread	90
S_IWUSR.....	359	thread ID.....	90
S_IXGRP	359	thread list.....	90
S_IXOTH	359	thread-safe.....	90
S_IXUSR	359	thread-safety.....	105
S_MSG.....	336	thread-specific data key	90
S_OUTPUT	336	tilde	91
S_RDBAND	336	timeb.....	366
S_RDNORM	336	timeouts.....	91
S_TYPEISMQ.....	360	timer	91
S_TYPEISSEM	360	timer overrun.....	91
S_TYPEISSHM	360	TIMER_ABSTIME.....	390
S_TYPEISTMO	360	TIMER_MAX.....	252
S_WRBAND	336	timer_t.....	368
S_WRNORM	336	timeval.....	347, 364
tab character	89	time_t	368
TABDLY	381	TMAGIC.....	378
TABn.....	381	TMAGLEN	378
TCIFLUSH	384	TMO	12
TCIOFF	384	TMPDIR.....	166
TCIOFLUSH	384	TMP_MAX	323
TCION	384	TMR.....	12
TCOOFF	384	TOEXEC	378
TCOON	384	token.....	91
TCP_NODELAY	285	TOREAD.....	378
TCSADRAIN	383	TOSTOP.....	383

TOWRITE.....	378	t_scalar_t	334
TPI.....	12	t_uscalar_t.....	334
TPP.....	12	UCHAR_MAX	259
TPS.....	12	ucontext_t.....	398
trace analyzer process.....	91	uid_t	368
trace controller process	91	UINTMAX_MAX.....	320
trace event	91	UINTN_MAX.....	319
trace event type.....	91	UINTPTR_MAX.....	320
trace event type mapping	92	UINT_FASTN_MAX.....	319
trace filter.....	92	UINT_LEASTN_MAX.....	319
trace generation version.....	92	UINT_MAX	259
trace log	92	ULLONG_MAX.....	260
trace point	92	ULONG_MAX	259
trace stream.....	92	UL_GETFSIZE.....	399
trace stream identifier.....	92	UL_SETFSIZE.....	399
trace system	92	unbind	93
traced process.....	92	undefined.....	6
TRACE_EVENT_NAME_MAX.....	252	unspecified	6
TRACE_NAME_MAX.....	252	UP.....	14
TRACE_SYS_MAX.....	252	upper multiplexing	85
TRACE_USER_EVENT_MAX.....	252	upshifting	93
tracing.....	27, 105	user database.....	94
tracing status of a trace stream	93	user ID.....	94
TRAP_BRKPT.....	307	user name	94
TRAP_TRACE.....	307	user trace event	94
TRC.....	12	USER_PROCESS.....	420
TRI	12	USHRT_MAX.....	259
TRL	13	utility.....	94, 109
TSA	13	Utility Syntax Guidelines.....	203
TSF.....	13	utmpx.....	420
TSGID.....	378	variable	95
TSH	13	variable assignment	109
TSP	13	VEOF	380
TSS	13	VEOL	380
TSUID.....	378	VERASE.....	380
TSVTX.....	378	vertical-tab character	95
TTY_NAME_MAX.....	252	VFS.....	362
TUEXEC.....	378	VINTR.....	380
TUREAD.....	378	VKILL.....	380
TUWRITE.....	378	VQUIT.....	380
TVERSION.....	378	VSTART	380
TVERSLEN.....	378	VSTOP.....	380
TYM.....	13	VSUSP.....	380
typed memory name space	93	VTDLY	382
typed memory object	93	VTn	382
typed memory pool.....	93	warning	
typed memory port	93	OB	9
TZ.....	166	OF	10
TZNAME_MAX.....	252	WCHAR_MAX	320, 424
T_FMT.....	246	WCHAR_MIN	320, 424
T_FMT_AMPM.....	246	WCONTINUED.....	374

WEOF	422, 424, 426
WEXITED	374
WEXITSTATUS	327, 374
white space	95
wide characters	119
wide-character code (C language)	95
wide-character input/output functions	95
wide-character string	95
WIFCONTINUED	374
WIFEXITED	327, 374
WIFSIGNALED	327, 374
WIFSTOPPED	327, 374
WINT_MAX	320
WINT_MIN	320
WNOHANG	327, 374
WNOWAIT	374
word	96
WORD_BIT	258-259
working directory	96
worldwide portability interface	96
WRDE_APPEND	428
WRDE_BADCHAR	428
WRDE_BADVAL	428
WRDE_CMDSUB	428
WRDE_DOOFFS	428
WRDE_NOCMD	428
WRDE_NOSPACE	428
WRDE_NOSYS	428
WRDE_REUSE	428
WRDE_SHOWERR	428
WRDE_SYNTAX	428
WRDE_UNDEF	428
write	96
WSTOPPED	374
WSTOPSIG	327, 374
WTERMSIG	327, 374
WUNTRACED	327, 374
W_OK	406
XSI	14 , 96
conformance	17, 21, 96
XSI options groups	24
XSI STREAMS	28
XSI system interfaces	
conformance	21
XSR	14
X_OK	406
YESEXPR	246
YESSTR	246
zombie process	96

IEEE Std 1003.1™, 2003 Edition

The Open Group Technical Standard
Base Specifications, Issue 6

Includes IEEE Std 1003.1™-2001 and IEEE Std 1003.1™-2001/Cor 1-2002

Information Technology — Portable Operating System Interface (POSIX®)

System Interfaces

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

and

The Open Group



THE *Open* GROUP

[This page intentionally left blank]

Abstract

This standard is simultaneously ISO/IEC 9945:2002, IEEE Std 1003.1-2001, and forms the core of the Single UNIX Specification, Version 3.

The IEEE Std 1003.1, 2003 Edition includes IEEE Std 1003.1-2001/Cor 1-2002 incorporated into IEEE Std 1003.1-2001 (base document). The Corrigendum addresses problems discovered since the approval of IEEE Std 1003.1-2001. These changes are mainly due to resolving integration issues raised by the merger of the base documents that were incorporated into IEEE Std 1003.1-2001, which is the single common revision to IEEE Std 1003.1™-1996, IEEE Std 1003.2™-1992, ISO/IEC 9945-1:1996, ISO/IEC 9945-2:1993, and the Base Specifications of The Open Group Single UNIX® Specification, Version 2.

This standard defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. This standard is intended to be used by both applications developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of this standard:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

This standard describes the external characteristics and facilities that are of importance to applications developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, input/output (I/O), job control, network, portable operating system interface (POSIX®), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

Copyright © 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. All rights reserved. This printing is by the International Organization for Standardization with special permission of the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. Published in Switzerland.

System Interfaces, Issue 6

Published 31 March 2003 by the Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, U.S.A.
ISBN: 0-7381-3436-8 PDF 0-7381-3564-X / SS95078 CD-ROM 0-7381-3563-1 / SE95078
Printed in the United States of America by the IEEE.

Published 31 March 2003 by The Open Group
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, U.K.
Document Number: C032
ISBN: 1-931624-24-0
Printed in the U.K. by The Open Group.

All rights reserved. No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission from both the IEEE and The Open Group.

Portions of this standard are derived with permission from copyrighted material owned by Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc.

Permissions

Authorization to photocopy portions of this standard for internal or personal use is granted provided that the appropriate fee is paid to the Copyright Clearance Center or the equivalent body outside of the U.S. Permission to make multiple copies for educational purposes in the U.S. requires agreement and a license fee to be paid to the Copyright Clearance Center.

Beyond these provisions, permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests to the copyright holders should be sent by email to austin-group-permissions@opengroup.org.

Feedback

This standard has been prepared by the Austin Group. Feedback relating to the material contained in this standard may be submitted using the Austin Group web site at <http://www.opengroup.org/austin/defectform.html>.

IEEE

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property, or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “AS IS”.

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with the IEEE.¹ Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, U.S.A.

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE Standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

A patent holder has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and non-discriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Authorization to photocopy portions of any individual standard for internal or personal use is granted in the U.S. by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to the Copyright Clearance Center.² Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center. To arrange for payment of the licensing fee, please contact:

Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923, U.S.A., Tel.: +1 978 750 8400

Amendments, corrigenda, and interpretations for this standard, or information about the IEEE standards development process, may be found at <http://standards.ieee.org>.

Full catalog and ordering information on all IEEE publications is available from the IEEE Online Catalog & Store at <http://shop.ieee.org/store>.

1. For this standard, please send comments via the Austin Group as requested on page iii.

2. Please refer to the special provisions for this standard on page iii concerning permissions from both copyright holders and arrangements to cover photocopying and reproduction across the world, as well as by commercial organizations wishing to license the material for use in product documentation.

The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The Open Group's mission is to offer all organizations concerned with open information infrastructures a forum to share knowledge, integrate open initiatives, and certify approved products and processes in a manner in which they continue to trust our impartiality.

In the global eCommerce world of today, no single economic entity can achieve independence while still ensuring interoperability. The assurance that products will interoperate with each other across differing systems and platforms is essential to the success of eCommerce and business workflow. The Open Group, with its proven testing and certification program, is the international guarantor of interoperability in the new century.

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group's members include some of the largest and most influential organizations in the world. The flexible structure of The Open Groups membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information is available on The Open Group web site at <http://www.opengroup.org>.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes the *Westwood* family of tests for this standard and the associated certification program for Version 3 of the Single UNIX Specification, as well tests for CDE, CORBA, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at <http://www.opengroup.org/testing>.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at <http://www.opengroup.org/pubs>.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at <http://www.opengroup.org/corrigenda>.

Full catalog and ordering information on all Open Group publications is available at <http://www.opengroup.org/pubs>.

Contents

Chapter 1	Introduction.....	1
1.1	Scope.....	1
1.2	Conformance	1
1.3	Normative References	1
1.4	Change History	1
1.5	Terminology	1
1.6	Definitions	3
1.7	Relationship to Other Formal Standards	3
1.8	Portability	3
1.8.1	Codes	3
1.9	Format of Entries.....	11
 Chapter 2	 General Information	 13
2.1	Use and Implementation of Functions	13
2.2	The Compilation Environment	13
2.2.1	POSIX.1 Symbols	13
2.2.1.1	The _POSIX_C_SOURCE Feature Test Macro	14
2.2.1.2	The _XOPEN_SOURCE Feature Test Macro.....	14
2.2.2	The Name Space.....	14
2.3	Error Numbers.....	21
2.3.1	Additional Error Numbers.....	28
2.4	Signal Concepts.....	28
2.4.1	Signal Generation and Delivery.....	28
2.4.2	Realtime Signal Generation and Delivery	29
2.4.3	Signal Actions	30
2.4.4	Signal Effects on Other Functions	34
2.5	Standard I/O Streams.....	34
2.5.1	Interaction of File Descriptors and Standard I/O Streams	35
2.5.2	Stream Orientation and Encoding Rules	36
2.6	STREAMS	38
2.6.1	Accessing STREAMS.....	39
2.7	XSI Interprocess Communication	39
2.7.1	IPC General Description.....	40
2.8	Realtime	41
2.8.1	Realtime Signals	41
2.8.2	Asynchronous I/O	41
2.8.3	Memory Management	43
2.8.3.1	Memory Locking.....	43
2.8.3.2	Memory Mapped Files.....	43
2.8.3.3	Memory Protection	43
2.8.3.4	Typed Memory Objects	44
2.8.4	Process Scheduling	44

2.8.5	Clocks and Timers	48
2.9	Threads.....	50
2.9.1	Thread-Safety.....	50
2.9.2	Thread IDs.....	51
2.9.3	Thread Mutexes.....	51
2.9.4	Thread Scheduling.....	52
2.9.5	Thread Cancellation	54
2.9.5.1	Cancelability States	54
2.9.5.2	Cancellation Points.....	55
2.9.5.3	Thread Cancellation Cleanup Handlers	57
2.9.5.4	Async-Cancel Safety.....	57
2.9.6	Thread Read-Write Locks.....	58
2.9.7	Thread Interactions with Regular File Operations	58
2.10	Sockets.....	58
2.10.1	Address Families.....	58
2.10.2	Addressing	59
2.10.3	Protocols	59
2.10.4	Routing.....	59
2.10.5	Interfaces.....	59
2.10.6	Socket Types.....	59
2.10.7	Socket I/O Mode.....	60
2.10.8	Socket Owner.....	60
2.10.9	Socket Queue Limits	60
2.10.10	Pending Error.....	60
2.10.11	Socket Receive Queue.....	61
2.10.12	Socket Out-of-Band Data State	61
2.10.13	Connection Indication Queue	62
2.10.14	Signals	62
2.10.15	Asynchronous Errors	62
2.10.16	Use of Options.....	63
2.10.17	Use of Sockets for Local UNIX Connections.....	66
2.10.17.1	Headers	66
2.10.18	Use of Sockets over Internet Protocols.....	66
2.10.19	Use of Sockets over Internet Protocols Based on IPv4.....	67
2.10.19.1	Headers	67
2.10.20	Use of Sockets over Internet Protocols Based on IPv6.....	67
2.10.20.1	Addressing	67
2.10.20.2	Compatibility with IPv4.....	68
2.10.20.3	Interface Identification.....	68
2.10.20.4	Options.....	69
2.10.20.5	Headers	70
2.11	Tracing.....	70
2.11.1	Tracing Data Definitions.....	71
2.11.1.1	Structures.....	71
2.11.1.2	Trace Stream Attributes.....	75
2.11.2	Trace Event Type Definitions	76
2.11.2.1	System Trace Event Type Definitions.....	76
2.11.2.2	User Trace Event Type Definitions.....	79

	2.11.3	Trace Functions.....	79
	2.12	Data Types.....	80
Chapter	3	System Interfaces	83
		Index.....	1669
 List of Tables			
	2-1	Value of Level for Socket Options.....	63
	2-2	Socket-Level Options.....	64
	2-3	Trace Option: System Trace Events.....	77
	2-4	Trace and Trace Event Filter Options: System Trace Events.....	78
	2-5	Trace and Trace Log Options: System Trace Events.....	78
	2-6	Trace, Trace Log, and Trace Event Filter Options: System Trace Events	79
	2-7	Trace Option: User Trace Event	79

Foreword

Structure of the Standard

This standard was originally developed by the Austin Group, a joint working group of members of the IEEE, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1, as one of the four volumes of IEEE Std 1003.1-2001. The standard was approved by ISO and IEC and published in four parts, correlating to the original volumes.

A mapping of the parts to the volumes is shown below:

ISO/IEC 9945 Part	IEEE Std 1003.1 Volume	Description
9945-1	Base Definitions	Includes general terms, concepts, and interfaces common to all parts of ISO/IEC 9945, including utility conventions and C-language header definitions.
9945-2	System Interfaces	Includes definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery.
9945-3	Shell and Utilities	Includes definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs.
9945-4	Rationale	Includes extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of ISO/IEC 9945 and why features were included or discarded by the standard developers.

All four parts comprise the entire standard, and are intended to be used together to accommodate significant internal referencing among them. POSIX-conforming systems are required to support all four parts.

Introduction

Note: This introduction is not part of IEEE Std 1003.1-2001, Standard for Information Technology — Portable Operating System Interface (POSIX).

This standard has been jointly developed by the IEEE and The Open Group. It is simultaneously an IEEE Standard, an ISO/IEC Standard, and an Open Group Technical Standard.

The Austin Group

This standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.³ The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group has been to revise, combine, and update the following standards: ISO/IEC 9945-1, ISO/IEC 9945-2, IEEE Std 1003.1, IEEE Std 1003.2, and the Base Specifications of The Open Group Single UNIX Specification.

After two initial meetings, an agreement was signed in July 1999 between The Open Group and the Institute of Electrical and Electronics Engineers (IEEE), Inc., to formalize the project with the first draft of the revised specifications being made available at the same time. Under this agreement, The Open Group and IEEE agreed to share joint copyright of the resulting work. The Open Group has provided the chair and secretariat for the Austin Group.

The base document for the revision was The Open Group's Base volumes of its Single UNIX Specification, Version 2. These were selected since they were a superset of the existing POSIX.1 and POSIX.2 specifications and had some organizational aspects that would benefit the audience for the new revision.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group's Technical Standard designation, and an ISO/IEC designation. This set of specifications forms the core of the Single UNIX Specification, Version 3.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see <http://www.opengroup.org/austin>.

3. The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

Background

The developers of this standard represent a cross section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, this standard describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application writers to write portable applications—it was developed with that goal in mind—it has been designated POSIX,⁴ an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of standards.

Audience

The intended audience for this standard is all persons concerned with an industry-wide standard operating system based on the UNIX system. This includes at least four groups of people:

1. Persons buying hardware and software systems
2. Persons managing companies that are deciding on future corporate computing directions
3. Persons implementing operating systems, and especially
4. Persons developing applications where portability is an objective

Purpose

Several principles guided the development of this standard:

- Application-Oriented

The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. This standard codifies the common, existing definition of the UNIX system.

- Interface, Not Implementation

This standard defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.

4. The name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

- Source, Not Object, Portability

This standard has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. This standard does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.

- The C Language

The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.

- No Superuser, No System Administration

There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in this standard. This standard is also not concerned with hardware constraints or system maintenance.

- Minimal Interface, Minimally Defined

In keeping with the historical design principles of the UNIX system, the mandatory core facilities of this standard have been kept as minimal as possible. Additional capabilities have been added as optional extensions.

- Broadly Implementable

The developers of this standard endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:

1. All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
2. Compatible systems that are not derived from the original UNIX system code
3. Emulations hosted on entirely different operating systems
4. Networked systems
5. Distributed systems
6. Systems running on a broad range of hardware

No direct references to this goal appear in this standard, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations

When the original version of IEEE Std 1003.1 was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and the subsequent revisions and addenda to all of them have consolidated this consensus, and this revision reflects the significantly increased level of consensus arrived at since the original versions. The earlier standards and their modifications specified a number of areas where consensus had not been reached before, and these are now reflected in this revision. The authors of the original versions tried, as much as possible, to follow the principles below

when creating new specifications:

1. By standardizing an interface like one in an historical implementation; for example, directories
2. By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended *tar* format defined in the *pax* utility
3. By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the *sigaction()* function

This revision tries to minimize the number of changes required to implementations which conform to the earlier versions of the approved standards to bring them into conformance with the current standard. Specifically, the scope of this work excluded doing any “new” work, but rather collecting into a single document what had been spread across a number of documents, and presenting it in what had been proven in practice to be a more effective way. Some changes to prior conforming implementations were unavoidable, primarily as a consequence of resolving conflicts found in prior revisions, or which became apparent when bringing the various pieces together.

However, since it references the 1999 version of the ISO C standard, and no longer supports “Common Usage C”, there are a number of unavoidable changes. Applications portability is similarly affected.

This standard is specifically not a codification of a particular vendor’s product.

It should be noted that implementations will have different kinds of extensions. Some will reflect “historical usage” and will be preserved for execution of pre-existing applications. These functions should be considered “obsolescent” and the standard functions used for new applications. Some extensions will represent functions beyond the scope of this standard. These need to be used with careful management to be able to adapt to future extensions of this standard and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code

A goal of this standard was to minimize additional work for the developers of applications. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

This Standard

This standard defines the Portable Operating System Interface (POSIX) requirements and consists of the following volumes:

- Base Definitions
- Shell and Utilities
- System Interfaces (this volume)
- Rationale (Informative)

This Volume

The System Interfaces volume describes the interfaces offered to application programs by POSIX-conformant systems. Readers are expected to be experienced C language programmers, and to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards.
- Chapter 2 contains important concepts, terms, and caveats relating to the rest of this volume.
- Chapter 3 defines the functional interfaces to the POSIX-conformant system.

Comprehensive references are available in the index.

Typographical Conventions

The following typographical conventions are used throughout this standard. In the text, this standard is referred to as IEEE Std 1003.1-2001, which is technically identical to The Open Group Base Specifications, Issue 6.

The typographical conventions listed here are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this standard.

Reference	Example	Notes
C-Language Data Structure	aiocb	
C-Language Data Structure Member	<i>aio_lio_opcode</i>	
C-Language Data Type	long	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	
C-Language Function Argument	<i>arg1</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	<sys/stat.h>	
C-Language Keyword	return	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	INET_ADDRSTRLEN	
C-Language Preprocessing Directive	#define	
Commands within a Utility	a, c	
Conversion Specification, Specifier / Modifier Character	<i>%A, g, E</i>	1
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	Hello, World	
Filename	/tmp	
Literal Character	<i>'c', '\r', '\'</i>	2
Literal String	<i>"abcde"</i>	2
Optional Items in Utility Syntax	[]	
Parameter	<i><directory pathname></i>	
Special Character	<i><newline></i>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	#include <sys/stat.h>	

Reference	Example	Notes
User Input and Example Code	echo Hello, World	5
Utility Name	awk	
Utility Operand	file_name	
Utility Option	-c	
Utility Option with Option-Argument	-w width	

Notes:

1. Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf* and *fscanf* formatting functions.
2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. For literal characters, `'\'` (or any of the other sequences such as `''`) is the same as the C constant `'\\'` (or `'\''`).
3. The style selected for some of the special characters, such as `<newline>`, matches the form of the input given to the *localed* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters `<tab>` or `<newline>`.
4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
5. Brackets shown in this font, `" [ ] "`, are part of the syntax and do *not* indicate optional items. In syntax the `' | '` symbol is used to separate alternatives, and ellipses (`" . . . "`) are used to show that additional arguments are optional.

Shading is used to identify extensions and options; see Section 1.8.1 (on page 3).

Footnotes and notes within the body of the normative text are for information only (informative).

Informative sections (such as Rationale, Change History, Application Usage, and so on) are denoted by continuous shading bars in the margins.

Ranges of values are indicated with parentheses or brackets as follows:

- (a,b) means the range of all values from a to b , including neither a nor b .
- $[a,b]$ means the range of all values from a to b , including a and b .
- $[a,b)$ means the range of all values from a to b , including a , but not b .
- $(a,b]$ means the range of all values from a to b , including b , but not a .

Notes:

1. Symbolic limits are used in this volume instead of fixed values for portability. The values of most of these constants are defined in the Base Definitions volume, **<limits.h>** or **<unistd.h>**.
2. The values of errors are defined in the Base Definitions volume, **<errno.h>**.

Participants

IEEE Std 1003.1-2001 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/SC22 WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair

Donald W. Cragun, Organizational Representative, IEEE PASC

Nicholas Stoughton, Organizational Representative, ISO/SC22 WG15

Mark Brown, Organizational Representative, The Open Group

Cathy Hughes, Technical Editor

Austin Group Technical Reviewers

Peter Anvin	Michael Gonzalez	Sandra O'Donnell
Bouazza Bachar	Joseph M. Gwinn	Frank Prindle
Theodore P. Baker	Jon Hitchcock	Curtis Royster Jr.
Walter Briscoe	Yvette Ho Sang	Glen Seeds
Mark Brown	Cathy Hughes	Keld Jorn Simonsen
Dave Butenhof	Lowell G. Johnson	Raja Srinivasan
Geoff Clare	Andrew Josey	Nicholas Stoughton
Donald W. Cragun	Michael Kavanaugh	Donn S. Terry
Lee Damico	David Korn	Fred Tydeman
Ulrich Drepper	Marc Aurele La France	Peter Van Der Veen
Paul Eggert	Jim Meyering	James Youngman
Joanna Farley	Gary Miller	Jim Zepeda
Clive D.W. Feather	Finnbarr P. Murphy	Jason Zions
Andrew Gollan	Joseph S. Myers	

Austin Group Working Group Members

Harold C. Adams
Peter Anvin
Pierre-Jean Arcos
Jay Ashford
Bouazza Bachar
Theodore P. Baker
Robert Barned
Joel Berman
David J. Blackwood
Shirley Bockstahler-Brandt
James Bottomley
Walter Briscoe
Andries Brouwer
Mark Brown
Eric W. Burger
Alan Burns
Andries Brouwer
Dave Butenhof
Keith Chow
Geoff Clare
Donald W. Cragun
Lee Damico
Juan Antonio De La Puente
Ming De Zhou
Steven J. Dovich
Richard P. Draves
Ulrich Drepper
Paul Eggert
Philip H. Enslow
Joanna Farley
Clive D.W. Feather
Pete Forman
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan

Michael Gonzalez
Karen D. Gordon
Joseph M. Gwinn
Steven A. Haaser
Charles E. Hammons
Chris J. Harding
Barry Hedquist
Vincent E. Henley
Karl Heubaum
Jon Hitchcock
Yvette Ho Sang
Niklas Holsti
Thomas Hosmer
Cathy Hughes
Jim D. Isaak
Lowell G. Johnson
Michael B. Jones
Andrew Josey
Michael J. Karels
Michael Kavanaugh
David Korn
Steven Kramer
Thomas M. Kurihara
Marc Aurele La France
C. Douglass Locke
Nick Maclaren
Roger J. Martin
Craig H. Meyer
Jim Meyering
Gary Miller
Finnbarr P. Murphy
Joseph S. Myers
John Napier
Peter E. Obermayer
James T. Oblinger

Sandra O'Donnell
Frank Prindle
Francois Riche
John D. Riley
Andrew K. Roach
Helmut Roth
Jaideep Roy
Curtis Royster Jr.
Stephen C. Schwarm
Glen Seeds
Richard Seibel
David L. Shroads Jr.
W. Olin Sibert
Keld Jorn Simonsen
Curtis Smith
Raja Srinivasan
Nicholas Stoughton
Marc J. Teller
Donn S. Terry
Fred Tydeman
Mark-Rene Uchida
Scott A. Valcourt
Peter Van Der Veen
Michael W. Vannier
Eric Vought
Frederick N. Webb
Paul A.T. Wolfgang
Garrett A. Wollman
James Youngman
Oren Yuen
Janusz Zalewski
Jim Zepeda
Jason Zions

The Open Group

When The Open Group approved the Base Specifications, Issue 6 on 12 September 2001, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair

Finnbarr P. Murphy, Vice-Chair

Mark Brown, Austin Group Liaison

Cathy Hughes, Technical Editor

Base Working Group Members

Bouazza Bachar

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Joanna Farley

Andrew Gollan

Karen D. Gordon

Gary Miller

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair
Joseph M. Gwinn, Vice-Chair
Jay Ashford, Functional Chair
Andrew Josey, Functional Chair
Curtis Royster Jr., Functional Chair
Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001. Balloters may have voted for approval, disapproval, or abstention:

Harold C. Adams	Steven A. Haaser	Frank Prindle
Pierre-Jean Arcos	Charles E. Hammons	Francois Riche
Jay Ashford	Chris J. Harding	John D. Riley
Theodore P. Baker	Barry Hedquist	Andrew K. Roach
Robert Barned	Vincent E. Henley	Helmut Roth
David J. Blackwood	Karl Heubaum	Jaideep Roy
Shirley Bockstahler-Brandt	Niklas Holsti	Curtis Royster Jr.
James Bottomley	Thomas Hosmer	Stephen C. Schwarm
Mark Brown	Jim D. Isaak	Richard Seibel
Eric W. Burger	Lowell G. Johnson	David L. Shroads Jr.
Alan Burns	Michael B. Jones	W. Olin Sibert
Dave Butenhof	Andrew Josey	Keld Jorn Simonsen
Keith Chow	Michael J. Karels	Nicholas Stoughton
Donald W. Cragun	Steven Kramer	Donn S. Terry
Juan Antonio De La Puente	Thomas M. Kurihara	Mark-Rene Uchida
Ming De Zhou	C. Douglass Locke	Scott A. Valcourt
Steven J. Dovich	Roger J. Martin	Michael W. Vannier
Richard P. Draves	Craig H. Meyer	Frederick N. Webb
Philip H. Enslow	Finnbarr P. Murphy	Paul A.T. Wolfgang
Michael Gonzalez	John Napier	Oren Yuen
Karen D. Gordon	Peter E. Obermayer	Janusz Zalewski
Joseph M. Gwinn	James T. Oblinger	

The following organizational representative voted on this standard:

Andrew Josey, X/Open Company Ltd.

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, it had the following membership:

Donald N. Heirman, Chair

James T. Carlo, Vice-Chair

Judith Gorman, Secretary

Satish K. Aggarwal

Mark D. Bowman

Gary R. Engmann

Harold E. Epstein

H. Landis Floyd

Jay Forster*

Howard M. Frazier

Ruben D. Garzon

James H. Gurney

Richard J. Holleman

Lowell G. Johnson

Robert J. Kennelly

Joseph L. Koepfinger*

Peter H. Lips

L. Bruce McClung

Daleep C. Mohla

James W. Moore

Robert F. Munzner

Ronald C. Petersen

Gerald H. Peterson

John B. Posey

Gary S. Robinson

Akio Tojo

Donald W. Zipse

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative

Donald R. Volzka, TAB Representative

Yvette Ho Sang, **Don Messina**, **Savoula Amanatidis**, IEEE Project Editors

* Member Emeritus

IEEE Std 1003.1-2001/Cor 1-2002 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/IEC JTC 1/SC22/WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair

Donald W. Cragun, Organizational Representative, IEEE PASC

Nicholas Stoughton, Organizational Representative, ISO/IEC JTC 1/SC22/WG15

Mark Brown, Organizational Representative, The Open Group

Cathy Fox, Technical Editor

Austin Group Technical Reviewers

Theodore P. Baker

Julian Blake

Andries Brouwer

Mark Brown

Dave Butenhof

Geoff Clare

Donald W. Cragun

Ken Dawson

Ulrich Drepper

Larry Dwyer

Paul Eggert

Joanna Farley

Clive D.W. Feather

Cathy Fox

Mark Funkenhauser

Lois Goldthwaite

Andrew Gollan

Michael Gonzalez

Bruno Haible

Ben Harris

Jon Hitchcock

Andreas Jaeger

Andrew Josey

Jonathan Lennox

Nick Maclaren

Jack McCann

Wilhelm Mueller

Joseph S. Myers

Frank Prindle

Kenneth Raeburn

Tim Robbins

Glen Seeds

Matthew Seitz

Keld Jorn Simonsen

Nicholas Stoughton

Alexander Terekhov

Donn S. Terry

Mike Wilson

Garrett A. Wollman

Mark Ziegast

Austin Group Working Group Members

Harold C. Adams
Alejandro Alonso
Jay Ashford
Theodore P. Baker
David J. Blackwood
Julian Blake
Mitchell Bonnett
Andries Brouwer
Mark Brown
Eric W. Burger
Alan Burns
Dave Butenhof
Keith Chow
Geoff Clare
Luis Cordova
Donald W. Cragun
Dragan Cvetkovic
Lee Damico
Ken Dawson
Jeroen Dekkers
Juan Antonio De La Puente
Steven J. Dovich
Ulrich Drepper
Dr. Sourav Dutta
Larry Dwyer
Paul Eggert
Joanna Farley

Clive D.W. Feather
Yaacov Fenster
Cathy Fox
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan
Michael Gonzalez
Karen D. Gordon
Scott Gudgel
Joseph M. Gwinn
Steven A. Haaser
Bruno Haible
Charles E. Hammons
Bryan Harold
Ben Harris
Barry Hedquist
Karl Heubaum
Jon Hitchcock
Andreas Jaeger
Andrew Josey
Kenneth Lang
Pi-Cheng Law
Jonathan Lennox
Nick Maclaren
Roger J. Martin
Jack McCann
George Miao

Wilhelm Mueller
Finnbarr P. Murphy
Joseph S. Myers
Alexey Neyman
Charles Ngethe
Peter Petrov
Frank Prindle
Vikram Punj
Kenneth Raeburn
Francois Riche
Tim Robbins
Curtis Royster Jr.
Diane Schleicher
Gil Shultz
Stephen C. Schwarm
Glen Seeds
Matthew Seitz
Keld Jorn Simonsen
Doug Stevenson
Nicholas Stoughton
Alexander Terekhov
Donn S. Terry
Mike Wilson
Garrett A. Wollman
Oren Yuen
Mark Ziegast

The Open Group

When The Open Group approved the Base Specifications, Issue 6, Technical Corrigendum 1 on 7 February 2003, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair

Finnbarr P. Murphy, Vice-Chair

Mark Brown, Austin Group Liaison

Cathy Fox, Technical Editor

Base Working Group Members

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Ulrich Drepper

Joanna Farley

Andrew Gollan

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001/Cor 1-2002 on 11 December 2002, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair

Joseph M. Gwinn, Vice-Chair

Jay Ashford, Functional Chair

Andrew Josey, Functional Chair

Curtis Royster Jr., Functional Chair

Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001/Cor 1-2002. Balloters may have voted for approval, disapproval, or abstention:

Alejandro Alonso

Jay Ashford

David J. Blackwood

Julian Blake

Mitchell Bonnett

Mark Brown

Dave Butenhof

Keith Chow

Luis Cordova

Donald W. Cragun

Steven J. Dovich

Dr. Sourav Dutta

Yaacov Fenster

Michael Gonzalez

Scott Gudgel

Charles E. Hammons

Bryan Harold

Barry Hedquist

Karl Heubaum

Lowell G. Johnson

Andrew Josey

Kenneth Lang

Pi-Cheng Law

George Miao

Roger J. Martin

Finnbarr P. Murphy

Charles Ngethe

Peter Petrov

Frank Prindle

Vikram Punj

Francois Riche

Curtis Royster Jr.

Diane Schleicher

Stephen C. Schwarm

Gil Shultz

Nicholas Stoughton

Donn S. Terry

Oren Yuen

Juan A. de la Puente

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 / Cor 1-2002 on 11 December 2002, the membership was as follows:

James T. Carlo, Chair

James H. Gurney, Vice-Chair

Judith Gorman, Secretary

Sid Bennett

H. Stephen Berger

Clyde R. Camp

Richard DeBlasio

Harold E. Epstein

Julian Forster*

Howard M. Frazier

Toshio Fukuda

Arnold M. Greenspan

Raymond Hapeman

Donald M. Heirman

Richard H. Hulett

Lowell G. Johnson

Joseph L. Koepfinger*

Peter H. Lips

Nader Mehravari

Daleep C. Mohla

William J. Moylan

Malcolm V. Thaden

Geoffrey O. Thompson

Howard L. Wolfman

Don Wright

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative

Satish K. Aggarwal, NRC Representative

Savoula Amanatidis, IEEE Standards Managing Editor

* Member Emeritus

Trademarks

The following information is given for the convenience of users of this standard and does not constitute endorsement of these products by The Open Group or the IEEE. There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

1003.1[™] is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

AIX[®] is a registered trademark of IBM Corporation.

AT&T[®] is a registered trademark of AT&T in the U.S.A. and other countries.

BSD[™] is a trademark of the University of California, Berkeley, U.S.A.

Hewlett-Packard[®], HP[®], and HP-UX[®] are registered trademarks of Hewlett-Packard Company.

IBM[®] is a registered trademark of International Business Machines Corporation.

The Open Group and Boundaryless Information Flow are trademarks and UNIX is a registered trademark of The Open Group in the United States and other countries. All other trademarks are the property of their respective owners.

POSIX[®] is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Sun[®] and Sun Microsystems[®] are registered trademarks of Sun Microsystems, Inc.

/usr/group[®] is a registered trademark of UniForum, the International Network of UNIX System Users.

Acknowledgements

The contributions of the following organizations to the development of IEEE Std 1003.1-2001 are gratefully acknowledged:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.
- The SC22 WG14 Committees.

This standard was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO SC22 WG15.

Referenced Documents

Normative References

Normative references for this standard are defined in the Base Definitions volume.

Informative References

The following documents are referenced in this standard:

1984 /usr/group Standard

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

Almasi and Gottlieb

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

ANSI C

American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.226-1994

American National Standard for Information Systems: Standard X3.226-1994, Programming Language Common LISP.

Brawer

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989, ISBN: 0-12-128470-0.

DeRemer and Pennello Article

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets*, SigPlan Notices, Volume 15, No. 8, August 1979.

Draft ANSI X3J11.1

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-1

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

FIPS 151-2

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

HP-UX Manual

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

IEC 60559: 1989

IEC 60559: 1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously designated IEC 559: 1989).

IEEE Std 754-1985

IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic.

IEEE Std 854-1987

IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.

- IEEE Std 1003.9-1992
IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.
- IETF RFC 791
Internet Protocol, Version 4 (IPv4), September 1981.
- IETF RFC 819
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982.
- IETF RFC 822
Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982.
- IETF RFC 919
Broadcasting Internet Datagrams, J. Mogul, October 1984.
- IETF RFC 920
Domain Requirements, J. Postel, J. Reynolds, October 1984.
- IETF RFC 921
Domain Name System Implementation Schedule, J. Postel, October 1984.
- IETF RFC 922
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984.
- IETF RFC 1034
Domain Names — Concepts and Facilities, P. Mockapetris, November 1987.
- IETF RFC 1035
Domain Names — Implementation and Specification, P. Mockapetris, November 1987.
- IETF RFC 1123
Requirements for Internet Hosts — Application and Support, R. Braden, October 1989.
- IETF RFC 1886
DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson, December 1995.
- IETF RFC 2045
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996.
- IETF RFC 2181
Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997.
- IETF RFC 2373
Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998.
- IETF RFC 2460
Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998.
- Internationalisation Guide
Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.
- ISO C (1990)
ISO/IEC 9899:1990, Programming Languages — C, including Amendment 1:1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).

ISO 2375:1985

ISO 2375:1985, Data Processing — Procedure for Registration of Escape Sequences.

ISO 8652:1987

ISO 8652:1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).

ISO/IEC 1539:1990

ISO/IEC 1539:1990, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

ISO/IEC 4873:1991

ISO/IEC 4873:1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.

ISO/IEC 6429:1992

ISO/IEC 6429:1992, Information Technology — Control Functions for Coded Character Sets.

ISO/IEC 6937:1994

ISO/IEC 6937:1994, Information Technology — Coded Character Set for Text Communication — Latin Alphabet.

ISO/IEC 8802-3:1996

ISO/IEC 8802-3:1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

ISO/IEC 8859

ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

Part 1: Latin Alphabet No. 1

Part 2: Latin Alphabet No. 2

Part 3: Latin Alphabet No. 3

Part 4: Latin Alphabet No. 4

Part 5: Latin/Cyrillic Alphabet

Part 6: Latin/Arabic Alphabet

Part 7: Latin/Greek Alphabet

Part 8: Latin/Hebrew Alphabet

Part 9: Latin Alphabet No. 5

Part 10: Latin Alphabet No. 6

Part 13: Latin Alphabet No. 7

Part 14: Latin Alphabet No. 8

Part 15: Latin Alphabet No. 9

ISO POSIX-1:1996

ISO/IEC 9945-1:1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Std 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.

ISO POSIX-2:1993

ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2-1992, as amended by ANSI/IEEE Std 1003.2a-1992).

Issue 1

X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

Issue 2

X/Open Portability Guide, January 1987:

- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
- Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)

Issue 3

X/Open Specification, 1988, 1989, February 1992:

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

Issue 4

CAE Specification, July 1992, published by The Open Group:

- System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
- Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
- System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

Issue 4, Version 2

CAE Specification, August 1994, published by The Open Group:

- System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

Issue 5

Technical Standard, February 1997, published by The Open Group:

- System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

Knuth Article

Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control, Volume 8, No. 6, October 1965.

KornShell

Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming Language*, March 1995, Prentice Hall.

MSE Working Draft

Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

POSIX.0: 1995

IEEE Std 1003.0-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical to ISO/IEC TR 14252).

POSIX.1: 1988

IEEE Std 1003.1-1988, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1: 1990

IEEE Std 1003.1-1990, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1a

P1003.1a, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — (C Language) Amendment.

POSIX.1d: 1999

IEEE Std 1003.1d-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 4: Additional Realtime Extensions [C Language].

POSIX.1g: 2000

IEEE Std 1003.1g-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 6: Protocol-Independent Interfaces (PII).

POSIX.1j: 2000

IEEE Std 1003.1j-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 5: Advanced Realtime Extensions [C Language].

POSIX.1q: 2000

IEEE Std 1003.1q-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 7: Tracing [C Language].

POSIX.2b

P1003.2b, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment.

POSIX.2d:-1994

IEEE Std 1003.2d-1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.

POSIX.13:-1998

IEEE Std 1003.13:1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.

Sarwate Article

Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications of the ACM, Volume 30, No. 8, August 1988.

Sprunt, Sha, and Lehoczky

Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*, The Journal of Real-Time Systems, Volume 1, 1989, Pages 27-60.

SVID, Issue 1

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.

SVID, Issue 2

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.

SVID, Issue 3

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.

The AWK Programming Language

Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.

UNIX Programmer's Manual

American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.

XNS, Issue 4

CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.

XNS, Issue 5

CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XNS, Issue 5.2

Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.

X/Open Curses, Issue 4, Version 2

CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.

Yacc

Yacc: Yet Another Compiler Compiler, Stephen C. Johnson, 1978.

Source Documents

Parts of the following documents were used to create the base documents for this standard:

AIX 3.2 Manual

AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).

OSF/1

OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).

OSF AES

Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

System V Release 2.0

- UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).
- UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).

System V Release 4.2

Operating System API Reference, UNIX SVR4.2 (1992) (ISBN: 0-13-017658-3).

Introduction

1.1 Scope

The scope of IEEE Std 1003.1-2001 is described in the Base Definitions volume of IEEE Std 1003.1-2001.

1.2 Conformance

Conformance requirements for IEEE Std 1003.1-2001 are defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance.

1.3 Normative References

Normative references for IEEE Std 1003.1-2001 are defined in the Base Definitions volume of IEEE Std 1003.1-2001.

1.4 Change History

Change history is described in the Rationale (Informative) volume of IEEE Std 1003.1-2001, and in the CHANGE HISTORY section of reference pages.

1.5 Terminology

This section appears in the Base Definitions volume of IEEE Std 1003.1-2001, but is repeated here for convenience:

For the purposes of IEEE Std 1003.1-2001, the following terminology definitions apply:

can

Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to IEEE Std 1003.1-2001. An application can rely on the existence of the feature or behavior.

implementation-defined

Describes a value or behavior that is not defined by IEEE Std 1003.1-2001 but is selected by an implementor. The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior so that it can be used correctly by an application.

legacy

Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable

applications. New applications should use alternative means of obtaining equivalent functionality.

may

Describes a feature or behavior that is optional for an implementation that conforms to IEEE Std 1003.1-2001. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

shall

For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or behavior that is mandatory. An application can rely on the existence of the feature or behavior.

For an application or user, describes a behavior that is mandatory.

should

For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or behavior that is recommended but not mandatory. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

For an application, describes a feature or behavior that is recommended programming practice for optimum portability.

undefined

Describes the nature of a value or behavior not defined by IEEE Std 1003.1-2001 which results from use of an invalid program construct or invalid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

unspecified

Describes the nature of a value or behavior not specified by IEEE Std 1003.1-2001 which results from use of a valid program construct or valid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

1.6 Definitions

Concepts and definitions are defined in the Base Definitions volume of IEEE Std 1003.1-2001.

1.7 Relationship to Other Formal Standards

Great care has been taken to ensure that this volume of IEEE Std 1003.1-2001 is fully aligned with the following standards:

ISO C (1999)

ISO/IEC 9899: 1999, Programming Languages — C.

Parts of the ISO/IEC 9899: 1999 standard (hereinafter referred to as the ISO C standard) are referenced to describe requirements also mandated by this volume of IEEE Std 1003.1-2001. Some functions and headers included within this volume of IEEE Std 1003.1-2001 have a version in the ISO C standard; in this case CX markings are added as appropriate to show where the ISO C standard has been extended (see Section 1.8.1). Any conflict between this volume of IEEE Std 1003.1-2001 and the ISO C standard is unintentional.

This volume of IEEE Std 1003.1-2001 also allows, but does not require, mathematics functions to support IEEE Std 754-1985 and IEEE Std 854-1987.

1.8 Portability

Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 and functions in the System Interfaces volume of IEEE Std 1003.1-2001 describe functionality that might not be fully portable to systems meeting the requirements for POSIX conformance (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance).

Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in the margin identifies the nature of the option, extension, or warning (see Section 1.8.1). For maximum portability, an application should avoid such functionality.

1.8.1 Codes

Margin codes and their meanings are listed in the Base Definitions volume of IEEE Std 1003.1-2001, but are repeated here for convenience:

ADV Advisory Information

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the ADV margin legend.

AIO Asynchronous Input and Output

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the AIO margin legend.

BAR Barriers

The functionality described is optional. The functionality described is also an extension to the

107		ISO C standard.
108		Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
109		Where additional semantics apply to a function, the material is identified by use of the BAR
110		margin legend.
111	BE	Batch Environment Services and Utilities
112		The functionality described is optional.
113		Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
114		Where additional semantics apply to a utility, the material is identified by use of the BE margin
115		legend.
116	CD	C-Language Development Utilities
117		The functionality described is optional.
118		Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
119		Where additional semantics apply to a utility, the material is identified by use of the CD margin
120		legend.
121	CPT	Process CPU-Time Clocks
122		The functionality described is optional. The functionality described is also an extension to the
123		ISO C standard.
124		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
125		Where additional semantics apply to a function, the material is identified by use of the CPT
126		margin legend.
127	CS	Clock Selection
128		The functionality described is optional. The functionality described is also an extension to the
129		ISO C standard.
130		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
131		Where additional semantics apply to a function, the material is identified by use of the CS
132		margin legend.
133	CX	Extension to the ISO C standard
134		The functionality described is an extension to the ISO C standard. Application writers may make
135		use of an extension as it is supported on all IEEE Std 1003.1-2001-conforming systems.
136		With each function or header from the ISO C standard, a statement to the effect that “any
137		conflict is unintentional” is included. That is intended to refer to a direct conflict.
138		IEEE Std 1003.1-2001 acts in part as a profile of the ISO C standard, and it may choose to further
139		constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered
140		conflicts.
141		Where additional semantics apply to a function or header, the material is identified by use of the
142		CX margin legend.
143	FD	FORTTRAN Development Utilities
144		The functionality described is optional.
145		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
146		Where additional semantics apply to a utility, the material is identified by use of the FD margin
147		legend.
148	FR	FORTTRAN Runtime Utilities
149		The functionality described is optional.

150		Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
151		Where additional semantics apply to a utility, the material is identified by use of the FR margin
152		legend.
153	FSC	File Synchronization
154		The functionality described is optional. The functionality described is also an extension to the
155		ISO C standard.
156		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
157		Where additional semantics apply to a function, the material is identified by use of the FSC
158		margin legend.
159	IP6	IPV6
160		The functionality described is optional. The functionality described is also an extension to the
161		ISO C standard.
162		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
163		Where additional semantics apply to a function, the material is identified by use of the IP6
164		margin legend.
165	MC1	Advisory Information and either Memory Mapped Files or Shared Memory Objects
166		The functionality described is optional. The functionality described is also an extension to the
167		ISO C standard.
168		This is a shorthand notation for combinations of multiple option codes.
169		Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
170		Where additional semantics apply to a function, the material is identified by use of the MC1
171		margin legend.
172		Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
173		Notation.
174	MC2	Memory Mapped Files, Shared Memory Objects, or Memory Protection
175		The functionality described is optional. The functionality described is also an extension to the
176		ISO C standard.
177		This is a shorthand notation for combinations of multiple option codes.
178		Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.
179		Where additional semantics apply to a function, the material is identified by use of the MC2
180		margin legend.
181		Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
182		Notation.
183	MC3	Memory Mapped Files, Shared Memory Objects, or Typed Memory Objects
184		The functionality described is optional. The functionality described is also an extension to the
185		ISO C standard.
186		This is a shorthand notation for combinations of multiple option codes.
187		Where applicable, functions are marked with the MC3 margin legend in the SYNOPSIS section.
188		Where additional semantics apply to a function, the material is identified by use of the MC3
189		margin legend.
190		Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
191		Notation.
192	MF	Memory Mapped Files
193		The functionality described is optional. The functionality described is also an extension to the

194		ISO C standard.
195		Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.
196		Where additional semantics apply to a function, the material is identified by use of the MF
197		margin legend.
198	ML	Process Memory Locking
199		The functionality described is optional. The functionality described is also an extension to the
200		ISO C standard.
201		Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
202		Where additional semantics apply to a function, the material is identified by use of the ML
203		margin legend.
204	MLR	Range Memory Locking
205		The functionality described is optional. The functionality described is also an extension to the
206		ISO C standard.
207		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
208		Where additional semantics apply to a function, the material is identified by use of the MLR
209		margin legend.
210	MON	Monotonic Clock
211		The functionality described is optional. The functionality described is also an extension to the
212		ISO C standard.
213		Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
214		Where additional semantics apply to a function, the material is identified by use of the MON
215		margin legend.
216	MPR	Memory Protection
217		The functionality described is optional. The functionality described is also an extension to the
218		ISO C standard.
219		Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
220		Where additional semantics apply to a function, the material is identified by use of the MPR
221		margin legend.
222	MSG	Message Passing
223		The functionality described is optional. The functionality described is also an extension to the
224		ISO C standard.
225		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
226		Where additional semantics apply to a function, the material is identified by use of the MSG
227		margin legend.
228	MX	IEC 60559 Floating-Point Option
229		The functionality described is optional. The functionality described is also an extension to the
230		ISO C standard.
231		Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.
232		Where additional semantics apply to a function, the material is identified by use of the MX
233		margin legend.
234	OB	Obsolescent
235		The functionality described may be withdrawn in a future version of this volume of
236		IEEE Std 1003.1-2001. Strictly Conforming POSIX Applications and Strictly Conforming XSI
237		Applications shall not use obsolescent features.

238		Where applicable, the material is identified by use of the OB margin legend.
239	OF	Output Format Incompletely Specified
240		The functionality described is an XSI extension. The format of the output produced by the utility
241		is not fully specified. It is therefore not possible to post-process this output in a consistent
242		fashion. Typical problems include unknown length of strings and unspecified field delimiters.
243		Where applicable, the material is identified by use of the OF margin legend.
244	OH	Optional Header
245		In the SYNOPSIS section of some interfaces in the System Interfaces volume of
246		IEEE Std 1003.1-2001 an included header is marked as in the following example:
247	OH	<pre>#include <sys/types.h></pre>
248		<pre>#include <grp.h></pre>
249		<pre>struct group *getgrnam(const char *name);</pre>
250		The OH margin legend indicates that the marked header is not required on XSI-conformant
251		systems.
252	PIO	Prioritized Input and Output
253		The functionality described is optional. The functionality described is also an extension to the
254		ISO C standard.
255		Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
256		Where additional semantics apply to a function, the material is identified by use of the PIO
257		margin legend.
258	PS	Process Scheduling
259		The functionality described is optional. The functionality described is also an extension to the
260		ISO C standard.
261		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
262		Where additional semantics apply to a function, the material is identified by use of the PS
263		margin legend.
264	RS	Raw Sockets
265		The functionality described is optional. The functionality described is also an extension to the
266		ISO C standard.
267		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
268		Where additional semantics apply to a function, the material is identified by use of the RS
269		margin legend.
270	RTS	Realtime Signals Extension
271		The functionality described is optional. The functionality described is also an extension to the
272		ISO C standard.
273		Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
274		Where additional semantics apply to a function, the material is identified by use of the RTS
275		margin legend.
276	SD	Software Development Utilities
277		The functionality described is optional.
278		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
279		Where additional semantics apply to a utility, the material is identified by use of the SD margin
280		legend.

281	SEM	Semaphores
282		The functionality described is optional. The functionality described is also an extension to the
283		ISO C standard.
284		Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
285		Where additional semantics apply to a function, the material is identified by use of the SEM
286		margin legend.
287	SHM	Shared Memory Objects
288		The functionality described is optional. The functionality described is also an extension to the
289		ISO C standard.
290		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
291		Where additional semantics apply to a function, the material is identified by use of the SHM
292		margin legend.
293	SIO	Synchronized Input and Output
294		The functionality described is optional. The functionality described is also an extension to the
295		ISO C standard.
296		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
297		Where additional semantics apply to a function, the material is identified by use of the SIO
298		margin legend.
299	SPI	Spin Locks
300		The functionality described is optional. The functionality described is also an extension to the
301		ISO C standard.
302		Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.
303		Where additional semantics apply to a function, the material is identified by use of the SPI
304		margin legend.
305	SPN	Spawn
306		The functionality described is optional. The functionality described is also an extension to the
307		ISO C standard.
308		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
309		Where additional semantics apply to a function, the material is identified by use of the SPN
310		margin legend.
311	SS	Process Sporadic Server
312		The functionality described is optional. The functionality described is also an extension to the
313		ISO C standard.
314		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
315		Where additional semantics apply to a function, the material is identified by use of the SS
316		margin legend.
317	TCT	Thread CPU-Time Clocks
318		The functionality described is optional. The functionality described is also an extension to the
319		ISO C standard.
320		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
321		Where additional semantics apply to a function, the material is identified by use of the TCT
322		margin legend.
323	TEF	Trace Event Filter
324		The functionality described is optional. The functionality described is also an extension to the
325		ISO C standard.

326		Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
327		Where additional semantics apply to a function, the material is identified by use of the TEF
328		margin legend.
329	THR	Threads
330		The functionality described is optional. The functionality described is also an extension to the
331		ISO C standard.
332		Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.
333		Where additional semantics apply to a function, the material is identified by use of the THR
334		margin legend.
335	TMO	Timeouts
336		The functionality described is optional. The functionality described is also an extension to the
337		ISO C standard.
338		Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
339		Where additional semantics apply to a function, the material is identified by use of the TMO
340		margin legend.
341	TMR	Timers
342		The functionality described is optional. The functionality described is also an extension to the
343		ISO C standard.
344		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
345		Where additional semantics apply to a function, the material is identified by use of the TMR
346		margin legend.
347	TPI	Thread Priority Inheritance
348		The functionality described is optional. The functionality described is also an extension to the
349		ISO C standard.
350		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
351		Where additional semantics apply to a function, the material is identified by use of the TPI
352		margin legend.
353	TPP	Thread Priority Protection
354		The functionality described is optional. The functionality described is also an extension to the
355		ISO C standard.
356		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
357		Where additional semantics apply to a function, the material is identified by use of the TPP
358		margin legend.
359	TPS	Thread Execution Scheduling
360		The functionality described is optional. The functionality described is also an extension to the
361		ISO C standard.
362		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
363		Where additional semantics apply to a function, the material is identified by use of the TPS
364		margin legend.
365	TRC	Trace
366		The functionality described is optional. The functionality described is also an extension to the
367		ISO C standard.
368		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
369		Where additional semantics apply to a function, the material is identified by use of the TRC
370		margin legend.

371	TRI	Trace Inherit
372		The functionality described is optional. The functionality described is also an extension to the
373		ISO C standard.
374		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
375		Where additional semantics apply to a function, the material is identified by use of the TRI
376		margin legend.
377	TRL	Trace Log
378		The functionality described is optional. The functionality described is also an extension to the
379		ISO C standard.
380		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
381		Where additional semantics apply to a function, the material is identified by use of the TRL
382		margin legend.
383	TSA	Thread Stack Address Attribute
384		The functionality described is optional. The functionality described is also an extension to the
385		ISO C standard.
386		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
387		Where additional semantics apply to a function, the material is identified by use of the TSA
388		margin legend.
389	TSF	Thread-Safe Functions
390		The functionality described is optional. The functionality described is also an extension to the
391		ISO C standard.
392		Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.
393		Where additional semantics apply to a function, the material is identified by use of the TSF
394		margin legend.
395	TSH	Thread Process-Shared Synchronization
396		The functionality described is optional. The functionality described is also an extension to the
397		ISO C standard.
398		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
399		Where additional semantics apply to a function, the material is identified by use of the TSH
400		margin legend.
401	TSP	Thread Sporadic Server
402		The functionality described is optional. The functionality described is also an extension to the
403		ISO C standard.
404		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
405		Where additional semantics apply to a function, the material is identified by use of the TSP
406		margin legend.
407	TSS	Thread Stack Size Attribute
408		The functionality described is optional. The functionality described is also an extension to the
409		ISO C standard.
410		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
411		Where additional semantics apply to a function, the material is identified by use of the TSS
412		margin legend.
413	TYM	Typed Memory Objects
414		The functionality described is optional. The functionality described is also an extension to the
415		ISO C standard.

Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TYM margin legend.

UP

User Portability Utilities

The functionality described is optional.

Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the UP margin legend.

XSI

Extension

The functionality described is an XSI extension. Functionality marked XSI is also an extension to the ISO C standard. Application writers may confidently make use of an extension on all systems supporting the X/Open System Interfaces Extension.

If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that reference page is an extension. See the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.439, XSI.

XSR

XSI STREAMS

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the XSR margin legend.

1.9 Format of Entries

The entries in Chapter 3 are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described. If it is necessary to include a header to use this function, the names of such headers are shown, for example:

```
#include <stdio.h>
```

DESCRIPTION

This section describes the functionality of the function or header.

RETURN VALUE

This section indicates the possible return values, if any.

If the implementation can detect errors, “successful completion” means that no error has been detected during execution of the function. If the implementation does detect an error, the error is indicated.

For functions where no errors are defined, “successful completion” means that if the implementation checks for errors, no error has been detected. If the implementation can detect errors, and an error is detected, the indicated return value is returned and *errno* may be set.

ERRORS

This section gives the symbolic names of the error values returned by a function or stored into a variable accessed through the symbol *errno* if an error occurs.

“No errors are defined” means that error values returned by a function or stored into a variable accessed through the symbol *errno*, if any, depend on the implementation.

EXAMPLES

This section is informative.

This section gives examples of usage, where appropriate. In the event of conflict between an example and a normative part of this volume of IEEE Std 1003.1-2001, the normative material is to be taken as correct.

APPLICATION USAGE

This section is informative.

This section gives warnings and advice to application writers about the entry. In the event of conflict between warnings and advice and a normative part of this volume of IEEE Std 1003.1-2001, the normative material is to be taken as correct.

RATIONALE

This section is informative.

This section contains historical information concerning the contents of this volume of IEEE Std 1003.1-2001 and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is informative.

This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

SEE ALSO

This section is informative.

This section gives references to related information.

CHANGE HISTORY

This section is informative.

This section shows the derivation of the entry and any significant changes that have been made to it.

General Information

This chapter covers information that is relevant to all the functions specified in Chapter 3 and the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

2.1 Use and Implementation of Functions

Each of the following statements shall apply unless explicitly stated otherwise in the detailed descriptions that follow:

1. If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behavior is undefined.
2. Any function declared in a header may also be implemented as a macro defined in the header, so a function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the left parenthesis that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a function even if it is also defined as a macro. The use of the C-language **#undef** construct to remove any such macro definition shall also ensure that an actual function is referred to.
3. Any invocation of a function that is implemented as a macro shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those function-like macros described in the following sections may be invoked in an expression anywhere a function with a compatible return type could be called.
4. Provided that a function can be declared without reference to any type defined in a header, it is also permissible to declare the function explicitly and use it without including its associated header.
5. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behavior is undefined.

2.2 The Compilation Environment

2.2.1 POSIX.1 Symbols

Certain symbols in this volume of IEEE Std 1003.1-2001 are defined in headers (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers). Some of those headers could also define symbols other than those defined by IEEE Std 1003.1-2001, potentially conflicting with symbols used by the application. Also, IEEE Std 1003.1-2001 defines symbols that are not permitted by other standards to appear in those headers without some control on the visibility of those symbols.

Symbols called “feature test macros” are used to control the visibility of symbols that might be included in a header. Implementations, future versions of IEEE Std 1003.1-2001, and other standards may define additional feature test macros.

In the compilation of an application that **#defines** a feature test macro specified by IEEE Std 1003.1-2001, no header defined by IEEE Std 1003.1-2001 shall be included prior to the definition of the feature test macro. This restriction also applies to any implementation-provided header in which these feature test macros are used. If the definition of the macro does not precede the **#include**, the result is undefined.

Feature test macros shall begin with the underscore character ('_').

2.2.1.1 The `_POSIX_C_SOURCE` Feature Test Macro

A POSIX-conforming application should ensure that the feature test macro `_POSIX_C_SOURCE` is defined before inclusion of any header.

When an application includes a header described by IEEE Std 1003.1-2001, and when this feature test macro is defined to have the value 200112L:

1. All symbols required by IEEE Std 1003.1-2001 to appear when the header is included shall be made visible.
2. Symbols that are explicitly permitted, but not required, by IEEE Std 1003.1-2001 to appear in that header (including those in reserved name spaces) may be made visible.
3. Additional symbols not required or explicitly permitted by IEEE Std 1003.1-2001 to be in that header shall not be made visible, except when enabled by another feature test macro.

Identifiers in IEEE Std 1003.1-2001 may only be undefined using the **#undef** directive as described in Section 2.1 (on page 13) or Section 2.2.2. These **#undef** directives shall follow all **#include** directives of any header in IEEE Std 1003.1-2001.

Note: The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been superseded by `_POSIX_C_SOURCE`.

2.2.1.2 The `_XOPEN_SOURCE` Feature Test Macro

An XSI-conforming application should ensure that the feature test macro `_XOPEN_SOURCE` is defined with the value 600 before inclusion of any header. This is needed to enable the functionality described in Section 2.2.1.1 and in addition to enable the XSI extension.

Since this volume of IEEE Std 1003.1-2001 is aligned with the ISO C standard, and since all functionality enabled by `_POSIX_C_SOURCE` set equal to 200112L is enabled by `_XOPEN_SOURCE` set equal to 600, there should be no need to define `_POSIX_C_SOURCE` if `_XOPEN_SOURCE` is so defined. Therefore, if `_XOPEN_SOURCE` is set equal to 600 and `_POSIX_C_SOURCE` is set equal to 200112L, the behavior is the same as if only `_XOPEN_SOURCE` is defined and set equal to 600. However, should `_POSIX_C_SOURCE` be set to a value greater than 200112L, the behavior is unspecified.

2.2.2 The Name Space

All identifiers in this volume of IEEE Std 1003.1-2001, except *environ*, are defined in at least one of the headers, as shown in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers. When `_XOPEN_SOURCE` or `_POSIX_C_SOURCE` is defined, each header defines or declares some identifiers, potentially conflicting with identifiers used by the application. The set of identifiers visible to the application consists of precisely those identifiers from the header pages of the included headers, as well as additional identifiers reserved for the implementation. In addition, some headers may make visible identifiers from other headers as indicated on the relevant header pages.

Implementations may also add members to a structure or union without controlling the visibility of those members with a feature test macro, as long as a user-defined macro with the same name cannot interfere with the correct interpretation of the program. The identifiers reserved for use by the implementation are described below:

1. Each identifier with external linkage described in the header section is reserved for use as an identifier with external linkage if the header is included.
2. Each macro described in the header section is reserved for any use if the header is included.
3. Each identifier with file scope described in the header section is reserved for use as an identifier with file scope in the same name space if the header is included.

The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by IEEE Std 1003.1-2001 and other POSIX standards. Implementations may add symbols to the headers shown in the following table, provided the identifiers for those symbols begin with the corresponding reserved prefixes in the following table, and do not use the reserved prefixes `posix_`, `POSIX_`, or `_POSIX_`.

		Header	Prefix	Suffix	Complete Name
584					
585					
586					
587	AIO	<aio.h>	aio_, lio_, AIO_, LIO_		
588		<arpa/inet.h>	in_, inet_		
589		<ctype.h>	to[a-z], is[a-z]		
590		<dirent.h>	d_		
591		<errno.h>	E[0-9], E[A-Z]		
592		<fcntl.h>	l_		
593		<glob.h>	gl_		
594		<grp.h>	gr_		
595		<inttypes.h>			int[0-9a-z]*_t, uint[0-9a-z]*_t
596					
597		<limits.h>		_MAX, _MIN	
598		<locale.h>	LC_[A-Z]		
599	MSG	<mqueue.h>	mq_, MQ_		
600	XSI	<ndbm.h>	dbm_		
601		<netdb.h>	h_, n_, p_, s_		
602		<net/if.h>	if_		
603		<netinet/in.h>	in_, ip_, s_, sin_		
604	IP6		in6_, s6_, sin6_		
605	XSI	<poll.h>	pd_, ph_, ps_		
606		<pthread.h>	pthread_, PTHREAD_		
607		<pwd.h>	pw_		
608		<regex.h>	re_, rm_		
609	PS	<sched.h>	sched_, SCHED_		
610	SEM	<semaphore.h>	sem_, SEM_		
611		<signal.h>	sa_, uc_, SIG[A-Z], SIG_[A-Z]		
612	XSI		ss_, sv_		
613	RTS		si_, SI_, sigev_, SIGEV_, sival_		
614	XSI	<stropts.h>	bi_, ic_, l_, sl_, str_		
615		<stdint.h>			int[0-9a-z]*_t, uint[0-9a-z]*_t
616					
617		<stdlib.h>	str[a-z]		
618		<string.h>	str[a-z], mem[a-z], wcs[a-z]		
619	XSI	<sys/ipc.h>	ipc_		key, pad, seq
620	MF	<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
621	XSI	<sys/msg.h>	msg		msg
622	XSI	<sys/resource.h>	rlim_, ru_		
623		<sys/select.h>	fd_, fds_, FD_		
624	XSI	<sys/sem.h>	sem		sem
625	XSI	<sys/shm.h>	shm		
626		<sys/socket.h>	ss_, sa_, if_, ifc_, ifru_, infu_, ifra_, msg_, cmsg_, l_		
627					
628		<sys/stat.h>	st_		
629	XSI	<sys/statvfs.h>	f_		
630		<sys/time.h>	fds_, it_, tv_, FD_		
631		<sys/times.h>	tms_		

Header	Prefix	Suffix	Complete Name
<sys/uio.h>	iov_		UIO_MAXIOV
<sys/un.h>	sun_		
<sys/utsname.h>	uts_		
<sys/wait.h>	si_, W[A-Z], P_		
<termios.h>	c_		
<time.h>	tm_		
	clock_, timer_, it_, tv_, CLOCK_, TIMER_		
<ucontext.h>	uc_, ss_		
<ulimit.h>	UL_		
<utime.h>	utim_		
<utmpx.h>	ut_	_LVL, _TIME, _PROCESS	
<wchar.h>	wcs[a-z]		
<wctype.h>	is[a-z], to[a-z]		
<wordexp.h>	we_		
ANY header		_t	

Note: The notation [A–Z] indicates any uppercase letter in the portable character set. The notation [a–z] indicates any lowercase letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name.

If any header in the following table is included, macros with the prefixes shown may be defined. After the last inclusion of a given header, an application may use identifiers with the corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the corresponding macro.

	Header	Prefix
660		
661		
662 XSI	<dlfcn.h>	RTLD_
663	<fcntl.h>	F_, O_, S_
664 XSI	<fmtmsg.h>	MM_
665	<fnmatch.h>	FNM_
666 XSI	<ftw.h>	FTW_
667	<glob.h>	GLOB_
668	<inttypes.h>	PRI[Xa-z], SCN[Xa-z]
669 XSI	<ndbm.h>	DBM_
670	<net/if.h>	IF_
671	<netinet/in.h>	IMPLINK_, IN_, INADDR_, IP_, IPPORT_, IPPROTO_, SOCK_
672 IP6		IPV6_, IN6_
673	<netinet/tcp.h>	TCP_
674 XSI	<nl_types.h>	NL_
675 XSI	<poll.h>	POLL_
676	<regex.h>	REG_
677	<signal.h>	SA_, SIG_[0-9a-z],
678 XSI		BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, SS_, SV_, TRAP_
679 XSI	<stropts.h>	FLUSH[A-Z], I_, M_, MUXID_R[A-Z], S_, SND[A-Z], STR_
680 XSI	<syslog.h>	LOG_
681 XSI	<sys/ipc.h>	IPC_
682 XSI	<sys/mman.h>	PROT_, MAP_, MS_
683 XSI	<sys/msg.h>	MSG[A-Z]
684 XSI	<sys/resource.h>	PRIO_, RLIM_, RLIMIT_, RUSAGE_
685 XSI	<sys/sem.h>	SEM_
686 XSI	<sys/shm.h>	SHM[A-Z], SHM_[A-Z]
687 XSI	<sys/socket.h>	AF_, CMSG_, MSG_, PF_, SCM_, SHUT_, SO_
688	<sys/stat.h>	S_
689 XSI	<sys/statvfs.h>	ST_
690 XSI	<sys/time.h>	FD_, ITIMER_
691 XSI	<sys/uio.h>	IOV_
692 XSI	<sys/wait.h>	BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, TRAP_
693	<termios.h>	V, I, O, TC, B[0-9] (See below.)
694	<wordexp.h>	WRDE_

695 The following are used to reserve complete names for the <stdint.h> header:

696 INT[0-9A-Za-z-]*_MIN
697 INT[0-9A-Za-z-]*_MAX
698 INT[0-9A-Za-z-]*_C
699 UINT[0-9A-Za-z-]*_MIN
700 UINT[0-9A-Za-z-]*_MAX
701 UINT[0-9A-Za-z-]*_C

702 **Note:** The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the
703 portable character set. The notation [0-9a-z-] indicates any digit, any lowercase letter in the
704 portable character set, or underscore.

705 The following reserved names are used as exact matches for **<termios.h>**:

706	XSI	CBAUD	EXTB	VDSUSP
707		DEFECHO	FLUSHO	VLNEXT
708		ECHOCTL	LOBLK	VREPRINT
709		ECHOKE	PENDIN	VSTATUS
710		ECHOPRT	SWTCH	VWERASE
711		EXTA	VDISCARD	

712 The following identifiers are reserved regardless of the inclusion of headers:

- 713 1. All identifiers that begin with an underscore and either an uppercase letter or another
714 underscore are always reserved for any use by the implementation.
- 715 2. All identifiers that begin with an underscore are always reserved for use as identifiers with
716 file scope in both the ordinary identifier and tag name spaces.
- 717 3. All identifiers in the table below are reserved for use as identifiers with external linkage.
718 Some of these identifiers do not appear in this volume of IEEE Std 1003.1-2001, but are
719 reserved for future use by the ISO C standard.

720	_Exit	ccoshf	csqrt	fputc	lrintl	sinh
721	abort	ccoshl	csqrtf	fputs	lround	sinhf
722	abs	ccosl	csqrtl	fputwc	lroundf	sinhl
723	acos	ceil	ctan	fputws	lroundl	sinl
724	acosf	ceilf	ctanf	fread	malloc	sprintf
725	acosh	ceilf	ctanl	free	mblen	sqrt
726	acoshf	ceill	ctgamma	freopen	mbrlen	sqrtf
727	acoshl	ceill	ctgammaf	frexp	mbrtowc	sqrtl
728	acosl	cerf	ctgamma	frexpf	mbsinit	srand
729	acosl	cerfc	ltime	frexpl	mbsrtowcs	sscanf
730	asctime	cerfcf	difftime	fscanf	mbstowcs	str[a-z]*
731	asin	cerfcl	div	fseek	mbtowc	strtof
732	asinf	cerff	erfcf	fsetpos	mem[a-z]*	strtoimax
733	asinh	cerfl	erfcl	ftell	mktime	strtold
734	asinhf	cexp1	erff	fwide	modf	strtoll
735	asinhl	cexp1f	erfl	fwprintf	modff	strtoull
736	asinl	cexp1l	errno	fwrite	modfl	strtoumax
737	asinl	cexp	exit	fwscanf	nan	swprintf
738	atan	cexp2	exp	getc	nanf	swscanf
739	atan2	cexp2f	exp2	getchar	nanl	system
740	atan2f	cexp2l	exp2f	getenv	nearbyint	tan
741	atan2l	cexpf	exp2l	gets	nearbyintf	tanf
742	atanf	cexpl	expf	getwc	nearbyintl	tanh
743	atanf	cimag	expl	getwchar	nextafterf	tanhf
744	atanh	cimagf	expm1	gmtime	nextafterl	tanhf
745	atanh	cimagl	expm1f	hypotf	nexttoward	tanl
746	atanhf	clearerr	expm1l	hypotl	nexttowardf	tgamma
747	atanhl	clgamma	fabs	ilogb	nexttowardl	tgammaf
748	atanl	clgammaf	fabsf	ilogbf	perror	tgammal
749	atanl	clgamma	fabsl	ilogbl	pow	time
750	atexit	clock	fclose	imaxabs	powf	tmpfile

751	atof	clog	fdim	imaxdiv	powl	tmpnam
752	atoi	clog10	fdimf	is[a-z]*	printf	to[a-z]*
753	atol	clog10f	fdiml	isblank	putc	trunc
754	atoll	clog10l	feclearexcept	iswblank	putchar	truncf
755	bsearch	clog1p	fegetenv	labs	puts	truncl
756	cabs	clog1pf	fegetexceptflag	ldexp	putwc	ungetc
757	cabsf	clog1pl	fegetround	ldexpf	putwchar	ungetwc
758	cabsl	clog2	feholdexcept	ldexpl	qsort	va_end
759	cacos	clog2f	feof	ldiv	raise	vfprintf
760	cacosf	clog2l	feraiseexcept	ldiv	rand	vfscanf
761	cacosh	clogf	ferror	lgammaf	realloc	vfwprintf
762	cacoshf	clogl	fesetenv	lgammal	remainderf	vfwscanf
763	cacoshl	conj	fesetexceptflag	llabs	remainderl	vprintf
764	cacosl	conjf	fesetround	llrint	remove	vscanf
765	calloc	conjl	fetestexcept	llrintf	remquo	vsprintf
766	carg	copysign	feupdateenv	llrintl	remquof	vsscanf
767	cargf	copysignf	fflush	llround	remquol	vswprintf
768	cargl	copysignl	fgetc	llroundf	rename	vswscanf
769	casin	cos	fgetpos	llroundl	rewind	vwprintf
770	casinf	cosf	fgets	localeconv	rint	vwscanf
771	casinh	cosh	fgetwc	localtime	rintf	wcrtomb
772	casinhf	coshf	fgetws	log	rintl	wcs[a-z]*
773	casinhl	coshl	floor	log10	round	wcstof
774	casinl	cosl	floorf	log10f	roundf	wcstoimax
775	catan	cpow	floorl	log10l	roundl	wcstold
776	catanf	cpowf	fma	log1p	scalbln	wcstoll
777	catanh	cpowl	fmaf	log1pf	scalblnf	wcstoull
778	catanh	cproj	fmal	log1pl	scalblnl	wcstoumax
779	catanhf	cprojf	fmax	log2	scalbn	wctob
780	catanhf	cprojl	fmaxf	log2f	scalbnf	wctomb
781	catanhl	creal	fmaxl	log2l	scalbnl	wctrans
782	catanhl	crealf	fmin	logb	scanf	wctype
783	catanl	creall	fminf	logbf	setbuf	wcwidth
784	cbrt	csin	fminl	logbl	setjmp	wmem[a-z]*
785	cbrtf	csinf	fmod	logf	setlocale	wprintf
786	cbrtl	csinh	fmodf	logl	setvbuf	wscanf
787	ccos	csinhf	fmodl	longjmp	signal	
788	ccosf	csinhl	fopen	lrint	sin	
789	ccosh	csinl	fprintf	lrintf	sinf	

Note: The notation [a-z] indicates any lowercase letter in the portable character set. The notation '*' indicates any combination of digits, letters in the portable character set, or underscore.

4. All functions and external identifiers defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers are reserved for use as identifiers with external linkage.
5. All the identifiers defined in this volume of IEEE Std 1003.1-2001 that have external linkage are always reserved for use as identifiers with external linkage.

No other identifiers are reserved.

Applications shall not declare or define identifiers with the same name as an identifier reserved in the same context. Since macro names are replaced whenever found, independent of scope and

name space, macro names matching any of the reserved identifier names shall not be defined by an application if any associated header is included.

Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`, headers may be included in any order, and each may be included more than once in a given scope, with no difference in effect from that of being included only once.

If used, the application shall ensure that a header is included outside of any external declaration or definition, and it shall be first included before the first reference to any type or macro it defines, or to any function or object it declares. However, if an identifier is declared or defined in more than one header, the second and subsequent associated headers may be included after the initial reference to the identifier. Prior to the inclusion of a header, the application shall not define any macros with names lexically identical to symbols defined by that header.

2.3 Error Numbers

Most functions can provide an error number. The means by which each function provides its error numbers is specified in its description.

Some functions provide the error number in a variable accessed through the symbol *errno*. The symbol *errno*, defined by including the `<errno.h>` header, expands to a modifiable lvalue of type **int**. It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.

The value of *errno* should only be examined when it is indicated to be valid by a function's return value. No function in this volume of IEEE Std 1003.1-2001 shall set *errno* to zero. For each thread of a process, the value of *errno* shall not be affected by function calls or assignments to *errno* by other threads.

Some functions return an error number directly as the function value. These functions return a value of zero to indicate success.

If more than one error occurs in processing a function call, any one of the possible errors may be returned, as the order of detection is undefined.

Implementations may support additional errors not included in this list, may generate errors included in this list under circumstances other than those described here, or may contain extensions or limitations that prevent some errors from occurring. The ERRORS section on each reference page specifies whether an error shall be returned, or whether it may be returned. Implementations shall not generate a different error number from the ones described here for error conditions described in this volume of IEEE Std 1003.1-2001, but may generate additional errors unless explicitly disallowed for a particular function.

Each implementation shall document, in the conformance document, situations in which each of the optional conditions defined in IEEE Std 1003.1-2001 is detected. The conformance document may also contain statements that one or more of the optional error conditions are not detected.

For functions under the Threads option for which `[EINTR]` is not listed as a possible error condition in this volume of IEEE Std 1003.1-2001, an implementation shall not return an error code of `[EINTR]`.

The following symbolic names identify the possible error numbers, in the context of the functions specifically defined in this volume of IEEE Std 1003.1-2001; these general descriptions are more precisely defined in the ERRORS sections of the functions that return them. Only these symbolic names should be used in programs, since the actual value of the error number is

unspecified. All values listed in this section shall be unique integer constant expressions with type **int** suitable for use in **#if** preprocessing directives, except as noted below. The values for all these names shall be found in the **<errno.h>** header defined in the Base Definitions volume of IEEE Std 1003.1-2001. The actual values are unspecified by this volume of IEEE Std 1003.1-2001.

[E2BIG]

Argument list too long. The sum of the number of bytes used by the new process image's argument list and environment list is greater than the system-imposed limit of {ARG_MAX} bytes.

or:

Lack of space in an output buffer.

or:

Argument is greater than the system-imposed maximum.

[EACCES]

Permission denied. An attempt was made to access a file in a way forbidden by its file access permissions.

[EADDRINUSE]

Address in use. The specified address is in use.

[EADDRNOTAVAIL]

Address not available. The specified address is not available from the local system.

[EAFNOSUPPORT]

Address family not supported. The implementation does not support the specified address family, or the specified address is not a valid address for the address family of the specified socket.

[EAGAIN]

Resource temporarily unavailable. This is a temporary condition and later calls to the same routine may complete normally.

[EALREADY]

Connection already in progress. A connection request is already in progress for the specified socket.

[EBADF]

Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a read (write) request is made to a file that is only open for writing (reading).

[EBADMSG]

Bad message. During a *read()*, *getmsg()*, *getpmsg()*, or *ioctl()* I_RECVFD request to a STREAMS device, a message arrived at the head of the STREAM that is inappropriate for the function receiving the message.

read() Message waiting to be read on a STREAM is not a data message.

getmsg() or *getpmsg()*

A file descriptor was received instead of a control message.

ioctl()

Control or data information was received instead of a file descriptor when I_RECVFD was specified.

or:

887	Bad Message. The implementation has detected a corrupted message.
888	[EBUSY]
889	Resource busy. An attempt was made to make use of a system resource that is not currently
890	available, as it is being used by another process in a manner that would have conflicted with
891	the request being made by this process.
892	[ECANCELED]
893	Operation canceled. The associated asynchronous operation was canceled before
894	completion.
895	[ECHILD]
896	No child process. A <i>wait()</i> or <i>waitpid()</i> function was executed by a process that had no
897	existing or unwaited-for child process.
898	[ECONNABORTED]
899	Connection aborted. The connection has been aborted.
900	[ECONNREFUSED]
901	Connection refused. An attempt to connect to a socket was refused because there was no
902	process listening or because the queue of connection requests was full and the underlying
903	protocol does not support retransmissions.
904	[ECONNRESET]
905	Connection reset. The connection was forcibly closed by the peer.
906	[EDEADLK]
907	Resource deadlock would occur. An attempt was made to lock a system resource that
908	would have resulted in a deadlock situation.
909	[DESTADDRREQ]
910	Destination address required. No bind address was established.
911	[EDOM]
912	Domain error. An input argument is outside the defined domain of the mathematical
913	function (defined in the ISO C standard).
914	[EDQUOT]
915	Reserved.
916	[EEXIST]
917	File exists. An existing file was mentioned in an inappropriate context; for example, as a
918	new link name in the <i>link()</i> function.
919	[EFAULT]
920	Bad address. The system detected an invalid address in attempting to use an argument of a
921	call. The reliable detection of this error cannot be guaranteed, and when not detected may
922	result in the generation of a signal, indicating an address violation, which is sent to the
923	process.
924	[EFBIG]
925	File too large. The size of a file would exceed the maximum file size of an implementation or
926	offset maximum established in the corresponding file description.
927	[EHOSTUNREACH]
928	Host is unreachable. The destination host cannot be reached (probably because the host is
929	down or a remote router cannot reach it).
930	[EIDRM]
931	Identifier removed. Returned during XSI interprocess communication if an identifier has

932	been removed from the system.
933	[EILSEQ]
934	Illegal byte sequence. A wide-character code has been detected that does not correspond to
935	a valid character, or a byte sequence does not form a valid wide-character code (defined in
936	the ISO C standard).
937	[EINPROGRESS]
938	Operation in progress. This code is used to indicate that an asynchronous operation has not
939	yet completed.
940	or:
941	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
942	immediately established.
943	[EINTR]
944	Interrupted function call. An asynchronous signal was caught by the process during the
945	execution of an interruptible function. If the signal handler performs a normal return, the
946	interrupted function call may return this condition (see the Base Definitions volume of
947	IEEE Std 1003.1-2001, <signal.h>).
948	[EINVAL]
949	Invalid argument. Some invalid argument was supplied; for example, specifying an
950	undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.
951	[EIO]
952	Input/output error. Some physical input or output error has occurred. This error may be
953	reported on a subsequent operation on the same file descriptor. Any other error-causing
954	operation on the same file descriptor may cause the [EIO] error indication to be lost.
955	[EISCONN]
956	Socket is connected. The specified socket is already connected.
957	[EISDIR]
958	Is a directory. An attempt was made to open a directory with write mode specified.
959	[ELOOP]
960	Symbolic link loop. A loop exists in symbolic links encountered during pathname
961	resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
962	are encountered during pathname resolution.
963	[EMFILE]
964	Too many open files. An attempt was made to open more than the maximum number of file
965	descriptors allowed in this process.
966	[EMLINK]
967	Too many links. An attempt was made to have the link count of a single file exceed
968	{LINK_MAX}.
969	[EMSGSIZE]
970	Message too large. A message sent on a transport provider was larger than an internal
971	message buffer or some other network limit.
972	or:
973	Inappropriate message buffer length.
974	[EMULTIHOP]
975	Reserved.

976	[ENAMETOOLONG]	
977		Filename too long. The length of a pathname exceeds {PATH_MAX}, or a pathname
978		component is longer than {NAME_MAX}. This error may also occur when pathname
979		substitution, as a result of encountering a symbolic link during pathname resolution, results
980		in a pathname string the size of which exceeds {PATH_MAX}.
981	[ENETDOWN]	
982		Network is down. The local network interface used to reach the destination is down.
983	[ENETRESET]	
984		The connection was aborted by the network.
985	[ENETUNREACH]	
986		Network unreachable. No route to the network is present.
987	[ENFILE]	
988		Too many files open in system. Too many files are currently open in the system. The system
989		has reached its predefined limit for simultaneously open files and temporarily cannot accept
990		requests to open another one.
991	[ENOBUFS]	
992		No buffer space available. Insufficient buffer resources were available in the system to
993		perform the socket operation.
994	XSR [ENODATA]	
995		No message available. No message is available on the STREAM head read queue.
996	[ENODEV]	
997		No such device. An attempt was made to apply an inappropriate function to a device; for
998		example, trying to read a write-only device such as a printer.
999	[ENOENT]	
1000		No such file or directory. A component of a specified pathname does not exist, or the
1001		pathname is an empty string.
1002	[ENOEXEC]	
1003		Executable file format error. A request is made to execute a file that, although it has the
1004		appropriate permissions, is not in the format required by the implementation for executable
1005		files.
1006	[ENOLCK]	
1007		No locks available. A system-imposed limit on the number of simultaneous file and record
1008		locks has been reached and no more are currently available.
1009	[ENOLINK]	
1010		Reserved.
1011	[ENOMEM]	
1012		Not enough space. The new process image requires more memory than is allowed by the
1013		hardware or system-imposed memory management constraints.
1014	[ENOMSG]	
1015		No message of the desired type. The message queue does not contain a message of the
1016		required type during XSI interprocess communication.
1017	[ENOPROTOPT]	
1018		Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the
1019		implementation.

1020		[ENOSPC]
1021		No space left on a device. During the <i>write()</i> function on a regular file or when extending a
1022		directory, there is no free space left on the device.
1023	XSR	[ENOSR]
1024		No STREAM resources. Insufficient STREAMS memory resources are available to perform a
1025		STREAMS-related function. This is a temporary condition; it may be recovered from if other
1026		processes release resources.
1027	XSR	[ENOSTR]
1028		Not a STREAM. A STREAM function was attempted on a file descriptor that was not
1029		associated with a STREAMS device.
1030		[ENOSYS]
1031		Function not implemented. An attempt was made to use a function that is not available in
1032		this implementation.
1033		[ENOTCONN]
1034		Socket not connected. The socket is not connected.
1035		[ENOTDIR]
1036		Not a directory. A component of the specified pathname exists, but it is not a directory,
1037		when a directory was expected.
1038		[ENOTEMPTY]
1039		Directory not empty. A directory other than an empty directory was supplied when an
1040		empty directory was expected.
1041		[ENOTSOCK]
1042		Not a socket. The file descriptor does not refer to a socket.
1043		[ENOTSUP]
1044		Not supported. The implementation does not support this feature of the Realtime Option
1045		Group.
1046		[ENOTTY]
1047		Inappropriate I/O control operation. A control function has been attempted for a file or
1048		special file for which the operation is inappropriate.
1049		[ENXIO]
1050		No such device or address. Input or output on a special file refers to a device that does not
1051		exist, or makes a request beyond the capabilities of the device. It may also occur when, for
1052		example, a tape drive is not on-line.
1053		[EOPNOTSUPP]
1054		Operation not supported on socket. The type of socket (address family or protocol) does not
1055		support the requested operation.
1056		[EOVERFLOW]
1057		Value too large to be stored in data type. An operation was attempted which would
1058		generate a value that is outside the range of values that can be represented in the relevant
1059		data type or that are allowed for a given data item.
1060		[EPERM]
1061		Operation not permitted. An attempt was made to perform an operation limited to
1062		processes with appropriate privileges or to the owner of a file or other resource.
1063		[EPIPE]
1064		Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process

1065	to read the data.
1066	[EPROTO]
1067	Protocol error. Some protocol error occurred. This error is device-specific, but is generally
1068	not related to a hardware failure.
1069	[EPROTONOSUPPORT]
1070	Protocol not supported. The protocol is not supported by the address family, or the protocol
1071	is not supported by the implementation.
1072	[EPROTOTYPE]
1073	Protocol wrong type for socket. The socket type is not supported by the protocol.
1074	[ERANGE]
1075	Result too large or too small. The result of the function is too large (overflow) or too small
1076	(underflow) to be represented in the available space (defined in the ISO C standard).
1077	[EROFS]
1078	Read-only file system. An attempt was made to modify a file or directory on a file system
1079	that is read-only.
1080	[ESPIPE]
1081	Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.
1082	[ESRCH]
1083	No such process. No process can be found corresponding to that specified by the given
1084	process ID.
1085	[ESTALE]
1086	Reserved.
1087	XSR [ETIME]
1088	
1089	
1090	
1091	
1092	[ETIMEDOUT]
1093	Connection timed out. The connection to a remote machine has timed out. If the connection
1094	timed out during execution of the function that reported this error (as opposed to timing
1095	out prior to the function being called), it is unspecified whether the function has completed
1096	some or all of the documented behavior associated with a successful completion of the
1097	function.
1098	or:
1099	Operation timed out. The time limit associated with the operation was exceeded before the
1100	operation completed.
1101	[ETXTBSY]
1102	Text file busy. An attempt was made to execute a pure-procedure program that is currently
1103	open for writing, or an attempt has been made to open for writing a pure-procedure
1104	program that is being executed.
1105	[EWOULDBLOCK]
1106	Operation would block. An operation on a socket marked as non-blocking has encountered
1107	a situation such as no data available that otherwise would have caused the function to
1108	suspend execution.

1109 A conforming implementation may assign the same values for [EWOULDBLOCK] and
1110 [EAGAIN].

1111 [EXDEV]

1112 Improper link. A link to a file on another file system was attempted.

1113 2.3.1 Additional Error Numbers

1114 Additional implementation-defined error numbers may be defined in `<errno.h>`.

1115 2.4 Signal Concepts

1116 2.4.1 Signal Generation and Delivery

1117 A signal is said to be “generated” for (or sent to) a process or thread when the event that causes
1118 the signal first occurs. Examples of such events include detection of hardware faults, timer
1119 RTS expiration, signals generated via the `sigevent` structure and terminal activity, as well as
1120 RTS invocations of the `kill()` and `sigqueue()` functions. In some circumstances, the same event
1121 generates signals for multiple processes.

1122 At the time of generation, a determination shall be made whether the signal has been generated
1123 for the process or for a specific thread within the process. Signals which are generated by some
1124 action attributable to a particular thread, such as a hardware fault, shall be generated for the
1125 thread that caused the signal to be generated. Signals that are generated in association with a
1126 process ID or process group ID or an asynchronous event, such as terminal activity, shall be
1127 generated for the process.

1128 Each process has an action to be taken in response to each signal defined by the system (see
1129 Section 2.4.3 (on page 30)). A signal is said to be “delivered” to a process when the appropriate
1130 action for the process and signal is taken. A signal is said to be “accepted” by a process when the
1131 signal is selected and returned by one of the `sigwait()` functions.

1132 During the time between the generation of a signal and its delivery or acceptance, the signal is
1133 said to be “pending”. Ordinarily, this interval cannot be detected by an application. However, a
1134 signal can be “blocked” from delivery to a thread. If the action associated with a blocked signal
1135 is anything other than to ignore the signal, and if that signal is generated for the thread, the
1136 signal shall remain pending until it is unblocked, it is accepted when it is selected and returned
1137 by a call to the `sigwait()` function, or the action associated with it is set to ignore the signal.
1138 Signals generated for the process shall be delivered to exactly one of those threads within the
1139 process which is in a call to a `sigwait()` function selecting that signal or has not blocked delivery
1140 of the signal. If there are no threads in a call to a `sigwait()` function selecting that signal, and if all
1141 threads within the process block delivery of the signal, the signal shall remain pending on the
1142 process until a thread calls a `sigwait()` function selecting that signal, a thread unblocks delivery
1143 of the signal, or the action associated with the signal is set to ignore the signal. If the action
1144 associated with a blocked signal is to ignore the signal and if that signal is generated for the
1145 process, it is unspecified whether the signal is discarded immediately upon generation or
1146 remains pending.

1147 Each thread has a “signal mask” that defines the set of signals currently blocked from delivery
1148 to it. The signal mask for a thread shall be initialized from that of its parent or creating thread,
1149 or from the corresponding thread in the parent process if the thread was created as the result of a
1150 call to `fork()`. The `pthread_sigmask()`, `sigaction()`, `sigprocmask()`, and `sigsuspend()` functions control
1151 the manipulation of the signal mask.

The determination of which action is taken in response to a signal is made at the time the signal is delivered, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated. If a subsequent occurrence of a pending signal is generated, it is implementation-defined as to whether the signal is delivered or accepted more than once in circumstances other than those in which queuing is required under the Realtime Signals Extension option. The order in which multiple, simultaneously pending signals outside the range SIGRTMIN to SIGRTMAX are delivered to or accepted by a process is unspecified.

When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any pending SIGCONT signals for that process shall be discarded. Conversely, when SIGCONT is generated for a process, all pending stop signals for that process shall be discarded. When SIGCONT is generated for a process that is stopped, the process shall be continued, even if the SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it shall remain pending until it is either unblocked or a stop signal is generated for the process.

An implementation shall document any condition not specified by this volume of IEEE Std 1003.1-2001 under which the implementation generates signals.

2.4.2 Realtime Signal Generation and Delivery

This section describes extensions to support realtime signal generation and delivery. This functionality is dependent on support of the Realtime Signals Extension option (and the rest of this section is not further shaded for this option).

Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O completion, interprocess message arrival, and the *sigqueue()* function, support the specification of an application-defined value, either explicitly as a parameter to the function or in a **sigevent** structure parameter. The **sigevent** structure is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	<i>sigev_notify</i>	Notification type.
int	<i>sigev_signo</i>	Signal number.
union signal	<i>sigev_value</i>	Signal value.
void(*)(unsigned signal)	<i>sigev_notify_function</i>	Notification function.
(pthread_attr_t*)	<i>sigev_notify_attributes</i>	Notification attributes.

The *sigev_notify* member specifies the notification mechanism to use when an asynchronous event occurs. This volume of IEEE Std 1003.1-2001 defines the following values for the *sigev_notify* member:

SIGEV_NONE	No asynchronous notification shall be delivered when the event of interest occurs.
SIGEV_SIGNAL	The signal specified in <i>sigev_signo</i> shall be generated for the process when the event of interest occurs. If the implementation supports the Realtime Signals Extension option and if the SA_SIGINFO flag is set for that signal number, then the signal shall be queued to the process and the value specified in <i>sigev_value</i> shall be the <i>si_value</i> component of the generated signal. If SA_SIGINFO is not set for that signal number, it is unspecified whether the signal is queued and what value, if any, is sent.
SIGEV_THREAD	A notification function shall be called to perform notification.

An implementation may define additional notification mechanisms.

The *sigev_signo* member specifies the signal to be generated. The *sigev_value* member is the application-defined value to be passed to the signal-catching function at the time of the signal delivery or to be returned at signal acceptance as the *si_value* member of the **siginfo_t** structure.

The **sigval** union is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	<i>sival_int</i>	Integer signal value.
void*	<i>sival_ptr</i>	Pointer signal value.

The *sival_int* member shall be used when the application-defined value is of type **int**; the *sival_ptr* member shall be used when the application-defined value is a pointer.

When a signal is generated by the *sigqueue()* function or any signal-generating function that supports the specification of an application-defined value, the signal shall be marked pending and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along with the application-specified signal value. Multiple occurrences of signals so generated are queued in FIFO order. It is unspecified whether signals so generated are queued when the SA_SIGINFO flag is not set for that signal.

Signals generated by the *kill()* function or other events that cause signals to occur, such as detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the implementation does not support queuing, shall have no effect on signals already queued for the same signal number.

When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the behavior shall be as if the implementation delivers the pending unblocked signal with the lowest signal number within that range. No other ordering of signal delivery is specified.

If, when a pending signal is delivered, there are additional signals queued to that signal number, the signal shall remain pending. Otherwise, the pending indication shall be reset.

Multi-threaded programs can use an alternate event notification mechanism. When a notification is processed, and the *sigev_notify* member of the **sigevent** structure has the value SIGEV_THREAD, the function *sigev_notify_function* is called with parameter *sigev_value*.

The function shall be executed in an environment as if it were the *start_routine* for a newly created thread with thread attributes specified by *sigev_notify_attributes*. If *sigev_notify_attributes* is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a *detachstate* attribute of PTHREAD_CREATE_JOINABLE results in undefined behavior. The signal mask of this thread is implementation-defined.

2.4.3 Signal Actions

There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN, or a pointer to a function. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior to entry of the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows:

SIG_DFL Signal-specific default action.

The default actions for the signals defined in this volume of IEEE Std 1003.1-2001 are specified under **<signal.h>**. If the Realtime Signals Extension option is supported, the default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX shall be to terminate the process abnormally.

1241		If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag. While a process is stopped, any additional signals that are sent to the process shall not be delivered until the process is continued, except SIGKILL which always terminates the receiving process. A process that is a member of an orphaned process group shall not be allowed to stop in response to the SIGTSTP, SIGTTIN, or SIGTTOU signals. In cases where delivery of one of these signals would stop such a process, the signal shall be discarded.	
1242			
1243			
1244			
1245			
1246			
1247			
1248			
1249			
1250		Setting a signal action to SIG_DFL for a signal that is pending, and whose default action is to ignore the signal (for example, SIGCHLD), shall cause the pending signal to be discarded, whether or not it is blocked. If the Realtime Signals	
1251			
1252	RTS	Extension option is supported, any queued values pending shall be discarded and the resources used to queue them shall be released and returned to the system for other use.	
1253			
1254			
1255			
1256		The default action for SIGCONT is to resume execution at the point where the process was stopped, after first handling any pending unblocked signals.	
1257			
1258	XSI	When a stopped process is continued, a SIGCHLD signal may be generated for its parent process, unless the parent process has set the SA_NOCLDSTOP flag.	
1259			
1260	SIG_IGN	Ignore signal.	
1261		Delivery of the signal shall have no effect on the process. The behavior of a process is undefined after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by <i>kill()</i> , <i>sigqueue()</i> , or <i>raise()</i> .	
1262	RTS		
1263	RTS		
1264		The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set to SIG_IGN.	
1265			
1266		Setting a signal action to SIG_IGN for a signal that is pending shall cause the pending signal to be discarded, whether or not it is blocked.	
1267			
1268		If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is unspecified, except as specified below.	
1269	XSI		
1270		If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and <i>wait()</i> , <i>waitid()</i> , and <i>waitpid()</i> shall fail and set <i>errno</i> to [ECHILD].	
1271			
1272			
1273			
1274			
1275			
1276	RTS	If the Realtime Signals Extension option is supported, any queued values pending shall be discarded and the resources used to queue them shall be released and made available to queue other signals.	
1277			
1278			
1279		<i>pointer to a function</i>	
1280		Catch signal.	
1281		On delivery of the signal, the receiving process is to execute the signal-catching function at the specified address. After returning from the signal-catching function, the receiving process shall resume execution at the point at which it was interrupted.	
1282			
1283			
1284			
1285		If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall be entered as a C-language function call as follows:	
1286			

1287 void *func*(int *signo*);

1288 XSI|RTS If the SA_SIGINFO flag for the signal is set, the signal-catching function shall be
1289 entered as a C-language function call as follows:

1290 void *func*(int *signo*, siginfo_t **info*, void **context*);

1291 where *func* is the specified signal-catching function, *signo* is the signal number of
1292 the signal being delivered, and *info* is a pointer to a **siginfo_t** structure defined in
1293 <signal.h> containing at least the following members:

1294		Member Type	Member Name	Description
1295		int	<i>si_signo</i>	Signal number.
1296		int	<i>si_code</i>	Cause of the signal.
1297		union sigval	<i>si_value</i>	Signal value.
1298				

1299 The *si_signo* member shall contain the signal number. This shall be the same as the
1300 *signo* parameter. The *si_code* member shall contain a code identifying the cause of
1301 the signal. The following values are defined for *si_code*:

1302 SI_USER The signal was sent by the *kill()* function. The implementation
1303 may set *si_code* to SI_USER if the signal was sent by the *raise()* or
1304 *abort()* functions or any similar functions provided as
1305 implementation extensions.

1306 RTS SI_QUEUE The signal was sent by the *sigqueue()* function.

1307 RTS SI_TIMER The signal was generated by the expiration of a timer set by
1308 *timer_settime()*.

1309 RTS SI_ASYNCIO The signal was generated by the completion of an asynchronous
1310 I/O request.

1311 RTS SI_MSGQ The signal was generated by the arrival of a message on an
1312 empty message queue.

1313 If the signal was not generated by one of the functions or events listed above, the
1314 *si_code* shall be set to an implementation-defined value that is not equal to any of
1315 the values defined above.

1316 RTS If the Realtime Signals Extension is supported, and *si_code* is one of SI_QUEUE,
1317 SI_TIMER, SI_ASYNCIO, or SI_MSGQ, then *si_value* shall contain the
1318 application-specified signal value. Otherwise, the contents of *si_value* are
1319 undefined.

1320 The behavior of a process is undefined after it returns normally from a signal-
1321 XSI catching function for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not
1322 RTS generated by *kill()*, *sigqueue()*, or *raise()*.

1323 The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

1324 If a process establishes a signal-catching function for the SIGCHLD signal while it
1325 has a terminated child process for which it has not waited, it is unspecified
1326 whether a SIGCHLD signal is generated to indicate that child process.

1327 When signal-catching functions are invoked asynchronously with process
1328 execution, the behavior of some of the functions defined by this volume of
1329 IEEE Std 1003.1-2001 is unspecified if they are called from a signal-catching
1330 function.

1331 The following table defines a set of functions that shall be either reentrant or non-
 1332 interruptible by signals and shall be async-signal-safe. Therefore applications may
 1333 invoke them, without restriction, from signal-catching functions:

1334	<code>_Exit()</code>	<code>fpathconf()</code>	<code>read()</code>	<code>sigset()</code>
1335	<code>_exit()</code>	<code>fstat()</code>	<code>readlink()</code>	<code>sigsuspend()</code>
1336	<code>abort()</code>	<code>fsync()</code>	<code>recv()</code>	<code>socket()</code>
1337	<code>accept()</code>	<code>ftruncate()</code>	<code>recvfrom()</code>	<code>socketpair()</code>
1338	<code>access()</code>	<code>getegid()</code>	<code>recvmsg()</code>	<code>stat()</code>
1339	<code>aio_error()</code>	<code>geteuid()</code>	<code>rename()</code>	<code>symlink()</code>
1340	<code>aio_return()</code>	<code>getgid()</code>	<code>rmdir()</code>	<code>sysconf()</code>
1341	<code>aio_suspend()</code>	<code>getgroups()</code>	<code>select()</code>	<code>tcdrain()</code>
1342	<code>alarm()</code>	<code>getpeername()</code>	<code>sem_post()</code>	<code>tcflow()</code>
1343	<code>bind()</code>	<code>getpgrp()</code>	<code>send()</code>	<code>tcflush()</code>
1344	<code>cfgetispeed()</code>	<code>getpid()</code>	<code>sendmsg()</code>	<code>tcgetattr()</code>
1345	<code>cfgetospeed()</code>	<code>getppid()</code>	<code>sendto()</code>	<code>tcgetpgrp()</code>
1346	<code>cfsetispeed()</code>	<code>getsockname()</code>	<code>setgid()</code>	<code>tcsendbreak()</code>
1347	<code>cfsetospeed()</code>	<code>getsockopt()</code>	<code>setpgid()</code>	<code>tcsetattr()</code>
1348	<code>chdir()</code>	<code>getuid()</code>	<code>setsid()</code>	<code>tcsetpgrp()</code>
1349	<code>chmod()</code>	<code>kill()</code>	<code>setsockopt()</code>	<code>time()</code>
1350	<code>chown()</code>	<code>link()</code>	<code>setuid()</code>	<code>timer_getoverrun()</code>
1351	<code>clock_gettime()</code>	<code>listen()</code>	<code>shutdown()</code>	<code>timer_gettime()</code>
1352	<code>close()</code>	<code>lseek()</code>	<code>sigaction()</code>	<code>timer_settime()</code>
1353	<code>connect()</code>	<code>lstat()</code>	<code>sigaddset()</code>	<code>times()</code>
1354	<code>creat()</code>	<code>mkdir()</code>	<code>sigdelset()</code>	<code>umask()</code>
1355	<code>dup()</code>	<code>mkfifo()</code>	<code>sigemptyset()</code>	<code>uname()</code>
1356	<code>dup2()</code>	<code>open()</code>	<code>sigfillset()</code>	<code>unlink()</code>
1357	<code>execle()</code>	<code>pathconf()</code>	<code>sigismember()</code>	<code>utime()</code>
1358	<code>execve()</code>	<code>pause()</code>	<code>sleep()</code>	<code>wait()</code>
1359	<code>fchmod()</code>	<code>pipe()</code>	<code>signal()</code>	<code>waitpid()</code>
1360	<code>fchown()</code>	<code>poll()</code>	<code>sigpause()</code>	<code>write()</code>
1361	<code>fcntl()</code>	<code>posix_trace_event()</code>	<code>sigpending()</code>	
1362	<code>fdatasync()</code>	<code>pselect()</code>	<code>sigprocmask()</code>	
1363	<code>fork()</code>	<code>raise()</code>	<code>sigqueue()</code>	

1364 All functions not in the above table are considered to be unsafe with respect to
 1365 signals. In the presence of signals, all functions defined by this volume of
 1366 IEEE Std 1003.1-2001 shall behave as defined when called from or interrupted by a
 1367 signal-catching function, with a single exception: when a signal interrupts an
 1368 unsafe function and the signal-catching function calls an unsafe function, the
 1369 behavior is undefined.

1370 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or
 1371 continue, the entire process shall be terminated, stopped, or continued, respectively.

2.4.4 Signal Effects on Other Functions

Signals affect the behavior of certain functions defined by this volume of IEEE Std 1003.1-2001 if delivered to a process while it is executing such a function. If the action of the signal is to terminate the process, the process shall be terminated and the function shall not return. If the action of the signal is to stop the process, the process shall stop until continued or terminated. Generation of a SIGCONT signal for the process shall cause the process to be continued, and the original function shall continue at the point the process was stopped. If the action of the signal is to invoke a signal-catching function, the signal-catching function shall be invoked; in this case the original function is said to be “interrupted” by the signal. If the signal-catching function executes a **return** statement, the behavior of the interrupted function shall be as described individually for that function, except as noted for unsafe functions. Signals that are ignored shall not affect the behavior of any function; signals that are blocked shall not affect the behavior of any function until they are unblocked and then delivered, except as specified for the *sigpending()* and *sigwait()* functions.

2.5 Standard I/O Streams

A stream is associated with an external file (which may be a physical device) by “opening” a file, which may involve “creating” a new file. Creating an existing file causes its former contents to be discarded if necessary. If a file can support positioning requests (such as a disk file, as opposed to a terminal), then a “file position indicator” associated with the stream is positioned at the start (byte number 0) of the file, unless the file is opened with append mode, in which case it is implementation-defined whether the file position indicator is initially positioned at the beginning or end of the file. The file position indicator is maintained by subsequent reads, writes, and positioning requests, to facilitate an orderly progression through the file. All input takes place as if bytes were read by successive calls to *fgetc()*; all output takes place as if bytes were written by successive calls to *fputc()*.

When a stream is “unbuffered”, bytes are intended to appear from the source or at the destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a block. When a stream is “fully buffered”, bytes are intended to be transmitted as a block when a buffer is filled. When a stream is “line buffered”, bytes are intended to be transmitted as a block when a newline byte is encountered. Furthermore, bytes are intended to be transmitted as a block when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line-buffered stream that requires the transmission of bytes. Support for these characteristics is implementation-defined, and may be affected via *setbuf()* and *setvbuf()*.

A file may be disassociated from a controlling stream by “closing” the file. Output streams are flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed (including the standard streams).

A file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all output streams are flushed) before program termination. Other paths to program termination, such as calling *abort()*, need not close all files properly.

The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE** object need not necessarily serve in place of the original.

At program start-up, three streams are predefined and need not be opened explicitly: *standard input* (for reading conventional input), *standard output* (for writing conventional output), and

1418 *standard error* (for writing diagnostic output). When opened, the standard error stream is not
 1419 fully buffered; the standard input and standard output streams are fully buffered if and only if
 1420 the stream can be determined not to refer to an interactive device.

1421 **2.5.1 Interaction of File Descriptors and Standard I/O Streams**

1422 CX This section describes the interaction of file descriptors and standard I/O streams. This
 1423 functionality is an extension to the ISO C standard (and the rest of this section is not further CX
 1424 shaded).

1425 An open file description may be accessed through a file descriptor, which is created using
 1426 functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as
 1427 *fopen()* or *popen()*. Either a file descriptor or a stream is called a “handle” on the open file
 1428 description to which it refers; an open file description may have several handles.

1429 Handles can be created or destroyed by explicit user action, without affecting the underlying
 1430 open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*,
 1431 and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

1432 A file descriptor that is never used in an operation that could affect the file offset (for example,
 1433 *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to one
 1434 (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include the
 1435 file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is not
 1436 used directly by the application to affect the file offset. The *read()* and *write()* functions
 1437 implicitly affect the file offset; *lseek()* explicitly affects it.

1438 The result of function calls involving any one handle (the “active handle”) is defined elsewhere
 1439 in this volume of IEEE Std 1003.1-2001, but if two or more handles are used, and any one of them
 1440 is a stream, the application shall ensure that their actions are coordinated as described below. If
 1441 this is not done, the result is undefined.

1442 A handle which is a stream is considered to be closed when either an *fclose()* or *freopen()* is
 1443 executed on it (the result of *freopen()* is a new stream, which cannot be a handle on the same
 1444 open file description as its previous value), or when the process owning that stream terminates
 1445 with *exit()*, *abort()*, or due to a signal. A file descriptor is closed by *close()*, *_exit()*, or the *exec*
 1446 functions when *FD_CLOEXEC* is set on that file descriptor.

1447 For a handle to become the active handle, the application shall ensure that the actions below are
 1448 performed between the last use of the handle (the current active handle) and the first use of the
 1449 second handle (the future active handle). The second handle then becomes the active handle. All
 1450 activity by the application affecting the file offset on the first handle shall be suspended until it
 1451 again becomes the active file handle. (If a stream function has as an underlying function one that
 1452 affects the file offset, the stream function shall be considered to affect the file offset.)

1453 The handles need not be in the same process for these rules to apply.

1454 Note that after a *fork()*, two handles exist where one existed before. The application shall ensure
 1455 that, if both handles can ever be accessed, they are both in a state where the other could become
 1456 the active handle first. The application shall prepare for a *fork()* exactly as if it were a change of
 1457 active handle. (If the only action performed by one of the processes is one of the *exec* functions or
 1458 *_exit()* (not *exit()*), the handle is never accessed in that process.)

1459 For the first handle, the first applicable condition below applies. After the actions required
 1460 below are taken, if the handle is still open, the application can close it.

- 1461 • If it is a file descriptor, no action is required.

- 1462 • If the only further action to be performed on any handle to this open file descriptor is to close
1463 it, no action need be taken.
- 1464 • If it is a stream which is unbuffered, no action need be taken.
- 1465 • If it is a stream which is line buffered, and the last byte written to the stream was a
1466 <newline> (that is, as if a:
1467

```
putc( '\n' )
```


1468 was the most recent operation on that stream), no action need be taken.
- 1469 • If it is a stream which is open for writing or appending (but not also open for reading), the
1470 application shall either perform an *fflush()*, or the stream shall be closed.
- 1471 • If the stream is open for reading and it is at the end of the file (*feof()* is true), no action need
1472 be taken.
- 1473 • If the stream is open with a mode that allows reading and the underlying open file
1474 description refers to a device that is capable of seeking, the application shall either perform
1475 an *fflush()*, or the stream shall be closed.
- 1476 Otherwise, the result is undefined.
- 1477 For the second handle:
- 1478 • If any previous active handle has been used by a function that explicitly changed the file
1479 offset, except as required above for the first handle, the application shall perform an *lseek()* or
1480 *fseek()* (as appropriate to the type of handle) to an appropriate location.
- 1481 If the active handle ceases to be accessible before the requirements on the first handle, above,
1482 have been met, the state of the open file description becomes undefined. This might occur during
1483 functions such as a *fork()* or *_exit()*.
- 1484 The *exec* functions make inaccessible all streams that are open at the time they are called,
1485 independent of which streams or file descriptors may be available to the new process image.
- 1486 When these rules are followed, regardless of the sequence of handles used, implementations
1487 shall ensure that an application, even one consisting of several processes, shall yield correct
1488 results: no data shall be lost or duplicated when writing, and all data shall be written in order,
1489 except as requested by seeks. It is implementation-defined whether, and under what conditions,
1490 all input is seen exactly once.
- 1491 If the rules above are not followed, the result is unspecified.
- 1492 Each function that operates on a stream is said to have zero or more “underlying functions”.
1493 This means that the stream function shares certain traits with the underlying functions, but does
1494 not require that there be any relation between the implementations of the stream function and its
1495 underlying functions.

1496 2.5.2 Stream Orientation and Encoding Rules

1497 For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an
1498 “orientation”. After a stream is associated with an external file, but before any operations are
1499 performed on it, the stream is without orientation. Once a wide-character input/output function
1500 has been applied to a stream without orientation, the stream shall become “wide-oriented”.
1501 Similarly, once a byte input/output function has been applied to a stream without orientation,
1502 the stream shall become “byte-oriented”. Only a call to the *freopen()* function or the *fwide()*
1503 function can otherwise alter the orientation of a stream.

1504 A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard*
 1505 *input*, *standard output*, and *standard error* shall be unoriented at program start-up.

1506 Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character
 1507 input/output functions cannot be applied to a byte-oriented stream. The remaining stream
 1508 operations shall not affect and shall not be affected by a stream's orientation, except for the
 1509 following additional restriction:

- 1510 • For wide-oriented streams, after a successful call to a file-positioning function that leaves the
 1511 file position indicator prior to the end-of-file, a wide-character output function can overwrite
 1512 a partial character; any file contents beyond the byte(s) written are henceforth undefined.

1513 Each wide-oriented stream has an associated **mbstate_t** object that stores the current parse state
 1514 of the stream. A successful call to *fgetpos()* shall store a representation of the value of this
 1515 **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to *fsetpos()* using
 1516 the same stored **fpos_t** value shall restore the value of the associated **mbstate_t** object as well as
 1517 the position within the controlled stream.

1518 Implementations that support multiple encoding rules associate an encoding rule with the
 1519 stream. The encoding rule shall be determined by the setting of the *LC_CTYPE* category in the
 1520 current locale at the time when the stream becomes wide-oriented. As with the stream's
 1521 orientation, the encoding rule associated with a stream cannot be changed once it has been set,
 1522 except by a successful call to *freopen()* which clears the encoding rule and resets the orientation
 1523 to unoriented.

1524 Although wide-oriented streams are conceptually sequences of wide characters, the external file
 1525 associated with a wide-oriented stream is a sequence of (possibly multi-byte) characters
 1526 generalized as follows:

- 1527 • Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte
 1528 encodings valid for use internal to the program).
- 1529 • A file need not begin nor end in the initial shift state.

1530 Moreover, the encodings used for characters may differ among files. Both the nature and choice
 1531 of such encodings are implementation-defined.

1532 The wide-character input functions read characters from the stream and convert them to wide
 1533 characters as if they were read by successive calls to the *fgetwc()* function. Each conversion shall
 1534 occur as if by a call to the *mbrtowc()* function, with the conversion state described by the stream's
 1535 own **mbstate_t** object, except the encoding rule associated with the stream is used instead of the
 1536 encoding rule implied by the *LC_CTYPE* category of the current locale.

1537 The wide-character output functions convert wide characters to (possibly multi-byte) characters
 1538 and write them to the stream as if they were written by successive calls to the *fputwc()* function.
 1539 Each conversion shall occur as if by a call to the *wcrtomb()* function, with the conversion state
 1540 described by the stream's own **mbstate_t** object, except the encoding rule associated with the
 1541 stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current
 1542 locale.

1543 An “encoding error” shall occur if the character sequence presented to the underlying *mbrtowc()*
 1544 function does not form a valid (generalized) character, or if the code value passed to the
 1545 underlying *wcrtomb()* function does not correspond to a valid (generalized) character. The
 1546 wide-character input/output functions and the byte input/output functions store the value of
 1547 the macro [EILSEQ] in *errno* if and only if an encoding error occurs.

2.6 STREAMS

STREAMS functionality is provided on implementations supporting the XSI STREAMS Option Group. This functionality is dependent on support of the XSI STREAMS option (and the rest of this section is not further shaded for this option).

STREAMS provides a uniform mechanism for implementing networking services and other character-based I/O. The STREAMS function provides direct access to protocol modules. STREAMS modules are unspecified objects. Access to STREAMS modules is provided by interfaces in IEEE Std 1003.1-2001. Creation of STREAMS modules is outside the scope of IEEE Std 1003.1-2001.

A STREAM is typically a full-duplex connection between a process and an open device or pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex connection between two processes. The STREAM itself exists entirely within the implementation and provides a general character I/O function for processes. It optionally includes one or more intermediate processing modules that are interposed between the process end of the STREAM (STREAM head) and a device driver at the end of the STREAM (STREAM end).

STREAMS I/O is based on messages. There are three types of message:

- *Data messages* containing actual data for input or output
- *Control data* containing instructions for the STREAMS modules and underlying implementation
- Other messages, which include file descriptors

The interface between the STREAM and the rest of the implementation is provided by a set of functions at the STREAM head. When a process calls *write()*, *writv()*, *putmsg()*, *putpmsg()*, or *ioctl()*, messages are sent down the STREAM, and *read()*, *readv()*, *getmsg()*, or *getpmsg()* accepts data from the STREAM and passes it to a process. Data intended for the device at the downstream end of the STREAM is packaged into messages and sent downstream, while data and signals from the device are composed into messages by the device driver and sent upstream to the STREAM head.

When a STREAMS-based device is opened, a STREAM shall be created that contains the STREAM head and the STREAM end (driver). If pipes are STREAMS-based in an implementation, when a pipe is created, two STREAMS shall be created, each containing a STREAM head. Other modules are added to the STREAM using *ioctl()*. New modules are “pushed” onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the STREAM was a push-down stack.

Priority

Message types are classified according to their queuing priority and may be *normal* (non-priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that determines its ordering when placed on a queue. Normal messages have a priority band of 0 and shall always be placed at the end of the queue following all other messages in the queue. High-priority messages are always placed at the head of a queue, but shall be discarded if there is already a high-priority message in the queue. Their priority band shall be ignored; they are high-priority by virtue of their type. Priority messages have a priority band greater than 0. Priority messages are always placed after any messages of the same or higher priority. High-priority and priority messages are used to send control and data information outside the normal flow of control. By convention, high-priority messages shall not be affected by flow control. Normal and priority messages have separate flow controls.

Message Parts

A process may access STREAMS messages that contain a data part, control part, or both. The data part is that information which is transmitted over the communication medium and the control information is used by the local STREAMS modules. The other types of messages are used between modules and are not accessible to processes. Messages containing only a data part are accessible via *putmsg()*, *putpmsg()*, *getmsg()*, *getpmsg()*, *read()*, *readv()*, *write()*, or *writev()*. Messages containing a control part with or without a data part are accessible via calls to *putmsg()*, *putpmsg()*, *getmsg()*, or *getpmsg()*.

2.6.1 Accessing STREAMS

A process accesses STREAMS-based files using the standard functions *close()*, *ioctl()*, *getmsg()*, *getpmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *putpmsg()*, *read()*, or *write()*. Refer to the applicable function definitions for general properties and errors.

Calls to *ioctl()* shall perform control functions on the STREAM associated with the file descriptor *fildev*. The control functions may be performed by the STREAM head, a STREAMS module, or the STREAMS driver for the STREAM.

STREAMS modules and drivers can detect errors, sending an error message to the STREAM head, thus causing subsequent functions to fail and set *errno* to the value specified in the message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl()* request alone by sending a negative acknowledgement message to the STREAM head. This shall cause just the pending *ioctl()* request to fail and set *errno* to the value specified in the message.

2.7 XSI Interprocess Communication

This section describes extensions to support interprocess communication. This functionality is dependent on support of the XSI extension (and the rest of this section is not further shaded for this option).

The following message passing, semaphore, and shared memory services form an XSI interprocess communication facility. Certain aspects of their operation are common, and are defined as follows.

IPC Functions		
<i>msgctl()</i>	<i>semctl()</i>	<i>shmctl()</i>
<i>msgget()</i>	<i>semget()</i>	<i>shmdt()</i>
<i>msgrcv()</i>	<i>semop()</i>	<i>shmget()</i>
<i>msgsnd()</i>	<i>shmat()</i>	

Another interprocess communication facility is provided by functions in the Realtime Option Group; see Section 2.8 (on page 41).

2.7.1 IPC General Description

Each individual shared memory segment, message queue, and semaphore set shall be identified by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a semaphore identifier, *semid*, and a message queue identifier, *msqid*. The identifiers shall be returned by calls to *shmget()*, *semget()*, and *msgget()*, respectively.

Associated with each identifier is a data structure which contains data related to the operations which may be or may have been performed; see the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/shm.h>**, **<sys/sem.h>**, and **<sys/msg.h>** for their descriptions.

Each of the data structures contains both ownership information and an **ipc_perm** structure (see the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/ipc.h>**) which are used in conjunction to determine whether or not read/write (read/alter for semaphores) permissions should be granted to processes using the IPC facilities. The *mode* member of the **ipc_perm** structure acts as a bit field which determines the permissions.

The values of the bits are given below in octal notation.

Bit	Meaning
0400	Read by user.
0200	Write by user.
0040	Read by group.
0020	Write by group.
0004	Read by others.
0002	Write by others.

The name of the **ipc_perm** structure is *shm_perm*, *sem_perm*, or *msg_perm*, depending on which service is being used. In each case, read and write/alter permissions shall be granted to a process if one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as appropriate):

- The process has appropriate privileges.
- The effective user ID of the process matches *xxx_perm.cuid* or *xxx_perm.uid* in the data structure associated with the IPC identifier, and the appropriate bit of the *user* field in *xxx_perm.mode* is set.
- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* but the effective group ID of the process matches *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, and the appropriate bit of the *group* field in *xxx_perm.mode* is set.
- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* and the effective group ID of the process does not match *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, but the appropriate bit of the *other* field in *xxx_perm.mode* is set.

Otherwise, the permission shall be denied.

2.8 Realtime

This section defines functions to support the source portability of applications with realtime requirements. The presence of many of these functions is dependent on support for implementation options described in the text.

The specific functional areas included in this section and their scope include the following. Full definitions of these terms can be found in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 3, Definitions.

- Semaphores
- Process Memory Locking
- Memory Mapped Files and Shared Memory Objects
- Priority Scheduling
- Realtime Signal Extension
- Timers
- Interprocess Communication
- Synchronized Input and Output
- Asynchronous Input and Output

All the realtime functions defined in this volume of IEEE Std 1003.1-2001 are portable, although some of the numeric parameters used by an implementation may have hardware dependencies.

2.8.1 Realtime Signals

RTS Realtime signal generation and delivery is dependent on support for the Realtime Signals Extension option.

See Section 2.4.2 (on page 29).

2.8.2 Asynchronous I/O

AIO The functionality described in this section is dependent on support of the Asynchronous Input and Output option (and the rest of this section is not further shaded for this option).

An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O functions. It is defined in the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>** and has at least the following members:

Member Type	Member Name	Description
int	<i>aio_fildes</i>	File descriptor.
off_t	<i>aio_offset</i>	File offset.
volatile void*	<i>aio_buf</i>	Location of buffer.
size_t	<i>aio_nbytes</i>	Length of transfer.
int	<i>aio_reqprio</i>	Request priority offset.
struct sigevent	<i>aio_sigevent</i>	Signal number and value.
int	<i>aio_lio_opcode</i>	Operation to be performed.

The *aio_fildes* element is the file descriptor on which the asynchronous operation is performed.

If **O_APPEND** is not set for the file descriptor *aio_fildes* and if *aio_fildes* is associated with a device that is capable of seeking, then the requested operation takes place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to the

operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to `SEEK_SET`. If `O_APPEND` is set for the file descriptor, or if *aio_fildes* is associated with a device that is incapable of seeking, write operations append to the file in the same order as the calls were made, with the following exception: under implementation-defined circumstances, such as operation on a multi-processor or when requests of differing priorities are submitted at the same time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming application to determine whether this relaxation applies, all strictly conforming applications which rely on ordering of output shall be written in such a way that they will operate correctly if the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified. The *aio_nbytes* and *aio_buf* elements are the same as the *nbyte* and *buf* arguments defined by *read()* and *write()*, respectively.

If `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, then asynchronous I/O is queued in priority order, with the priority of each asynchronous operation based on the current scheduling priority of the calling process. The *aio_reqprio* member can be used to lower (but not raise) the asynchronous I/O operation priority and is within the range zero through `{AIO_PRIO_DELTA_MAX}`, inclusive. Unless both `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing asynchronous I/O requests is unspecified. When both `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing of requests submitted by processes whose schedulers are not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is unspecified. The priority of an asynchronous request is computed as (process scheduling priority) minus *aio_reqprio*. The priority assigned to each asynchronous I/O request is an indication of the desired order of execution of the request relative to other asynchronous I/O requests for this file. If `_POSIX_PRIORITIZED_IO` is defined, requests issued with the same priority to a character special file are processed by the underlying device in FIFO order; the order of processing of requests of the same priority issued to files that are not character special files is unspecified. Numerically higher priority values indicate requests of higher priority. The value of *aio_reqprio* has no effect on process scheduling priority. When prioritized asynchronous I/O requests to the same file are blocked waiting for a resource required for that I/O operation, the higher-priority I/O requests shall be granted the resource before lower-priority I/O requests are granted the resource. The relative priority of asynchronous I/O and synchronous I/O is implementation-defined. If `_POSIX_PRIORITIZED_IO` is defined, the implementation shall define for which files I/O prioritization is supported.

The *aio_sigevent* determines how the calling process shall be notified upon I/O completion, as specified in Section 2.4.1 (on page 28). If *aio_sigevent.sigev_notify* is `SIGEV_NONE`, then no signal shall be posted upon I/O completion, but the error status for the operation and the return status for the operation shall be set appropriately.

The *aio_lio_opcode* field is used only by the *lio_listio()* call. The *lio_listio()* call allows multiple asynchronous I/O operations to be submitted at a single time. The function takes as an argument an array of pointers to **aiocb** structures. Each **aiocb** structure indicates the operation to be performed (read or write) via the *aio_lio_opcode* field.

The address of the **aiocb** structure is used as a handle for retrieving the error status and return status of the asynchronous operation while it is in progress.

The **aiocb** structure and the data buffers associated with the asynchronous I/O operation are being used by the system for asynchronous I/O while, and only while, the error status of the asynchronous operation is equal to `[EINPROGRESS]`. Applications shall not modify the **aiocb** structure while the structure is being used by the system for asynchronous I/O.

The return status of the asynchronous operation is the number of bytes transferred by the I/O operation. If the error status is set to indicate an error completion, then the return status is set to

1755 the return value that the corresponding *read()*, *write()*, or *fsync()* call would have returned.
 1756 When the error status is not equal to [EINPROGRESS], the return status shall reflect the return
 1757 status of the corresponding synchronous operation.

1758 **2.8.3 Memory Management**

1759 2.8.3.1 *Memory Locking*

1760 MLR Range memory locking operations are defined in terms of pages. Implementations may restrict |
 1761 the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes,
 1762 is the value of the configurable system variable {PAGESIZE}. If an implementation has no
 1763 restrictions on size or alignment, it may specify a 1-byte page size.

1764 ML|MLR Memory locking guarantees the residence of portions of the address space. It is |
 1765 implementation-defined whether locking memory guarantees fixed translation between virtual
 1766 addresses (as seen by the process) and physical addresses. Per-process memory locks are not
 1767 inherited across a *fork()*, and all memory locks owned by a process are unlocked upon *exec* or
 1768 process termination. Unmapping of an address range removes any memory locks established on
 1769 that address range by this process.

1770 2.8.3.2 *Memory Mapped Files*

1771 MF The functionality described in this section is dependent on support of the Memory Mapped Files
 1772 option (and the rest of this section is not further shaded for this option).

1773 Range memory mapping operations are defined in terms of pages. Implementations may
 1774 restrict the size and alignment of range mappings to be on page-size boundaries. The page size,
 1775 in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has
 1776 no restrictions on size or alignment, it may specify a 1-byte page size.

1777 Memory mapped files provide a mechanism that allows a process to access files by directly
 1778 incorporating file data into its address space. Once a file is mapped into a process address space,
 1779 the data can be manipulated as memory. If more than one process maps a file, its contents are
 1780 shared among them. If the mappings allow shared write access, then data written into the
 1781 memory object through the address space of one process appears in the address spaces of all
 1782 processes that similarly map the same portion of the memory object.

1783 SHM Shared memory objects are named regions of storage that may be independent of the file system
 1784 and can be mapped into the address space of one or more processes to allow them to share the
 1785 associated memory.

1786 SHM An *unlink()* of a file or *shm_unlink()* of a shared memory object, while causing the removal of the
 1787 name, does not unmap any mappings established for the object. Once the name has been
 1788 removed, the contents of the memory object are preserved as long as it is referenced. The
 1789 memory object remains referenced as long as a process has the memory object open or has some
 1790 area of the memory object mapped.

1791 2.8.3.3 *Memory Protection*

1792 MPR MF The functionality described in this section is dependent on support of the Memory Protection
 1793 and Memory Mapped Files option (and the rest of this section is not further shaded for these
 1794 options).

1795 When an object is mapped, various application accesses to the mapped region may result in
 1796 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and
 1797 SIGSEGV is used to indicate a protection violation or misuse of an address:

- 1798 • A mapping may be restricted to disallow some types of access.
- 1799 • Write attempts to memory that was mapped without write access, or any access to memory
- 1800 mapped PROT_NONE, shall result in a SIGSEGV signal.
- 1801 • References to unmapped addresses shall result in a SIGSEGV signal.
- 1802 • Reference to whole pages within the mapping, but beyond the current length of the object,
- 1803 shall result in a SIGBUS signal.
- 1804 • The size of the object is unaffected by access beyond the end of the object (even if a SIGBUS is
- 1805 not generated).

1806 2.8.3.4 *Typed Memory Objects*

1807 TYM The functionality described in this section is dependent on support of the Typed Memory
 1808 Objects option (and the rest of this section is not further shaded for this option).

1809 Implementations may support the Typed Memory Objects option without supporting the
 1810 Memory Mapped Files option or the Shared Memory Objects option. Typed memory objects are
 1811 implementation-configurable named storage pools accessible from one or more processors in a
 1812 system, each via one or more ports, such as backplane buses, LANs, I/O channels, and so on.
 1813 Each valid combination of a storage pool and a port is identified through a name that is defined
 1814 at system configuration time, in an implementation-defined manner; the name may be
 1815 independent of the file system. Using this name, a typed memory object can be opened and
 1816 mapped into process address space. For a given storage pool and port, it is necessary to support
 1817 both dynamic allocation from the pool as well as mapping at an application-supplied offset
 1818 within the pool; when dynamic allocation has been performed, subsequent deallocation must be
 1819 supported. Lastly, accessing typed memory objects from different ports requires a method for
 1820 obtaining the offset and length of contiguous storage of a region of typed memory (dynamically
 1821 allocated or not); this allows typed memory to be shared among processes and/or processors
 1822 while being accessed from the desired port.

1823 2.8.4 **Process Scheduling**

1824 PS The functionality described in this section is dependent on support of the Process Scheduling
 1825 option (and the rest of this section is not further shaded for this option).

1826 **Scheduling Policies**

1827 The scheduling semantics described in this volume of IEEE Std 1003.1-2001 are defined in terms
 1828 of a conceptual model that contains a set of thread lists. No implementation structures are
 1829 necessarily implied by the use of this conceptual model. It is assumed that no time elapses
 1830 during operations described using this model, and therefore no simultaneous operations are
 1831 possible. This model discusses only processor scheduling for runnable threads, but it should be
 1832 noted that greatly enhanced predictability of realtime applications results if the sequencing of
 1833 other resources takes processor scheduling policy into account.

1834 There is, conceptually, one thread list for each priority. A runnable thread will be on the thread
 1835 list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty
 1836 thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose
 1837 of a scheduling policy is to define the allowable operations on this set of lists (for example,
 1838 moving threads between and within lists).

1839 Each process shall be controlled by an associated scheduling policy and priority. These
 1840 parameters may be specified by explicit application execution of the *sched_setscheduler()* or
 1841 *sched_setparam()* functions.

Each thread shall be controlled by an associated scheduling policy and priority. These parameters may be specified by explicit application execution of the *pthread_setschedparam()* function.

Associated with each policy is a priority range. Each policy definition shall specify the minimum priority range for that policy. The priority ranges for each policy may but need not overlap the priority ranges of other policies.

A conforming implementation shall select the thread that is defined as being at the head of the highest priority non-empty thread list to become a running thread, regardless of its associated policy. This thread is then removed from its thread list.

Four scheduling policies are specifically required. Other implementation-defined scheduling policies may be defined. The following symbols are defined in the Base Definitions volume of IEEE Std 1003.1-2001, <**sched.h**>:

SCHED_FIFO First in, first out (FIFO) scheduling policy.

SCHED_RR Round robin scheduling policy.

SCHED_SPORADIC Sporadic server scheduling policy.

SCHED_OTHER Another scheduling policy.

The values of these symbols shall be distinct.

SCHED_FIFO

Conforming implementations shall include a scheduling policy called the FIFO scheduling policy.

Threads scheduled under this policy are chosen from a thread list that is ordered by the time its threads have been on the list without being executed; generally, the head of the list is the thread that has been on the list the longest time, and the tail is the thread that has been on the list the shortest time.

Under the **SCHED_FIFO** policy, the modification of the definitional thread lists is as follows:

1. When a running thread becomes a preempted thread, it becomes the head of the thread list for its priority.
2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list for its priority.
3. When a running thread calls the *sched_setscheduler()* function, the process specified in the function call is modified to the specified policy and the priority specified by the *param* argument.
4. When a running thread calls the *sched_setparam()* function, the priority of the process specified in the function call is modified to the priority specified by the *param* argument.
5. When a running thread calls the *pthread_setschedparam()* function, the thread specified in the function call is modified to the specified policy and the priority specified by the *param* argument.
6. When a running thread calls the *pthread_setschedprio()* function, the thread specified in the function call is modified to the priority specified by the *prio* argument.
7. If a thread whose policy or priority has been modified other than by *pthread_setschedprio()* is a running thread or is runnable, it then becomes the tail of the thread list for its new priority.

8. If a thread whose policy or priority has been modified by *pthread_setschedprio()* is a running thread or is runnable, the effect on its position in the thread list depends on the direction of the modification, as follows:
 - a. If the priority is raised, the thread becomes the tail of the thread list.
 - b. If the priority is unchanged, the thread does not change position in the thread list.
 - c. If the priority is lowered, the thread becomes the head of the thread list.
9. When a running thread issues the *sched_yield()* function, the thread becomes the tail of the thread list for its priority.
10. At no other time is the position of a thread with this scheduling policy within the thread lists affected.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when *SCHED_FIFO* is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 priorities for this policy.

SCHED_RR

Conforming implementations shall include a scheduling policy called the “round robin” scheduling policy. This policy shall be identical to the *SCHED_FIFO* policy with the additional condition that when the implementation detects that a running thread has been executing as a running thread for a time period of the length returned by the *sched_rr_get_interval()* function or longer, the thread shall become the tail of its thread list and the head of that thread list shall be removed and made a running thread.

The effect of this policy is to ensure that if there are multiple *SCHED_RR* threads at the same priority, one of them does not monopolize the processor. An application should not rely only on the use of *SCHED_RR* to ensure application progress among multiple threads if the application includes threads using the *SCHED_FIFO* policy at the same or higher priority levels or *SCHED_RR* threads at a higher priority level.

A thread under this policy that is preempted and subsequently resumes execution as a running thread completes the unexpired portion of its round robin interval time period.

For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when *SCHED_RR* is provided as the parameter. Conforming implementations shall provide a priority range of at least 32 priorities for this policy.

SCHED_SPORADIC

The functionality described in this section is dependent on support of the Process Sporadic Server or Thread Sporadic Server options (and the rest of this section is not further shaded for these options).

If *_POSIX_SPORADIC_SERVER* or *_POSIX_THREAD_SPORADIC_SERVER* is defined, the implementation shall include a scheduling policy identified by the value *SCHED_SPORADIC*.

The sporadic server policy is based primarily on two parameters: the replenishment period and the available execution capacity. The replenishment period is given by the *sched_ss_repl_period* member of the **sched_param** structure. The available execution capacity is initialized to the value given by the *sched_ss_init_budget* member of the same parameter. The sporadic server policy is identical to the *SCHED_FIFO* policy with some additional conditions that cause the thread’s assigned priority to be switched between the values specified by the *sched_priority* and

1928 *sched_ss_low_priority* members of the **sched_param** structure.

1929 The priority assigned to a thread using the sporadic server scheduling policy is determined in
 1930 the following manner: if the available execution capacity is greater than zero and the number of
 1931 pending replenishment operations is strictly less than *sched_ss_max_repl*, the thread is assigned
 1932 the priority specified by *sched_priority*; otherwise, the assigned priority shall be
 1933 *sched_ss_low_priority*. If the value of *sched_priority* is less than or equal to the value of
 1934 *sched_ss_low_priority*, the results are undefined. When active, the thread shall belong to the
 1935 thread list corresponding to its assigned priority level, according to the mentioned priority
 1936 assignment. The modification of the available execution capacity and, consequently of the
 1937 assigned priority, is done as follows:

- 1938 1. When the thread at the head of the *sched_priority* list becomes a running thread, its
 1939 execution time shall be limited to at most its available execution capacity, plus the
 1940 resolution of the execution time clock used for this scheduling policy. This resolution shall
 1941 be implementation-defined.
- 1942 2. Each time the thread is inserted at the tail of the list associated with *sched_priority*—
 1943 because as a blocked thread it became runnable with priority *sched_priority* or because a
 1944 replenishment operation was performed—the time at which this operation is done is
 1945 posted as the *activation_time*.
- 1946 3. When the running thread with assigned priority equal to *sched_priority* becomes a
 1947 preempted thread, it becomes the head of the thread list for its priority, and the execution
 1948 time consumed is subtracted from the available execution capacity. If the available
 1949 execution capacity would become negative by this operation, it shall be set to zero.
- 1950 4. When the running thread with assigned priority equal to *sched_priority* becomes a blocked
 1951 thread, the execution time consumed is subtracted from the available execution capacity,
 1952 and a replenishment operation is scheduled, as described in 6 and 7. If the available
 1953 execution capacity would become negative by this operation, it shall be set to zero.
- 1954 5. When the running thread with assigned priority equal to *sched_priority* reaches the limit
 1955 imposed on its execution time, it becomes the tail of the thread list for
 1956 *sched_ss_low_priority*, the execution time consumed is subtracted from the available
 1957 execution capacity (which becomes zero), and a replenishment operation is scheduled, as
 1958 described in 6 and 7.
- 1959 6. Each time a replenishment operation is scheduled, the amount of execution capacity to be
 1960 replenished, *replenish_amount*, is set equal to the execution time consumed by the thread
 1961 since the *activation_time*. The replenishment is scheduled to occur at *activation_time* plus
 1962 *sched_ss_repl_period*. If the scheduled time obtained is before the current time, the
 1963 replenishment operation is carried out immediately. Several replenishment operations may
 1964 be pending at the same time, each of which will be serviced at its respective scheduled
 1965 time. With the above rules, the number of replenishment operations simultaneously
 1966 pending for a given thread that is scheduled under the sporadic server policy shall not be
 1967 greater than *sched_ss_max_repl*.
- 1968 7. A replenishment operation consists of adding the corresponding *replenish_amount* to the
 1969 available execution capacity at the scheduled time. If, as a consequence of this operation,
 1970 the execution capacity would become larger than *sched_ss_initial_budget*, it shall be
 1971 rounded down to a value equal to *sched_ss_initial_budget*. Additionally, if the thread was
 1972 runnable or running, and had assigned priority equal to *sched_ss_low_priority*, then it
 1973 becomes the tail of the thread list for *sched_priority*.

1974 Execution time is defined in Section 2.2.2 (on page 14).

1975 For this policy, changing the value of a CPU-time clock via `clock_settime()` shall have no effect on
1976 its behavior.

1977 For this policy, valid priorities shall be within the range returned by the `sched_get_priority_min()`
1978 and `sched_get_priority_max()` functions when `SCHED_SPORADIC` is provided as the parameter.
1979 Conforming implementations shall provide a priority range of at least 32 distinct priorities for
1980 this policy.

1981 **SCHED_OTHER**

1982 Conforming implementations shall include one scheduling policy identified as `SCHED_OTHER`
1983 (which may execute identically with either the FIFO or round robin scheduling policy). The
1984 effect of scheduling threads with the `SCHED_OTHER` policy in a system in which other threads
1985 ss are executing under `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is implementation-
1986 defined.

1987 This policy is defined to allow strictly conforming applications to be able to indicate in a
1988 portable manner that they no longer need a realtime scheduling policy.

1989 For threads executing under this policy, the implementation shall use only priorities within the
1990 range returned by the `sched_get_priority_max()` and `sched_get_priority_min()` functions when
1991 `SCHED_OTHER` is provided as the parameter.

1992 **2.8.5 Clocks and Timers**

1993 TMR The functionality described in this section is dependent on support of the Timers option (and the
1994 rest of this section is not further shaded for this option).

1995 The `<time.h>` header defines the types and manifest constants used by the timing facility.

1996 **Time Value Specification Structures**

1997 Many of the timing facility functions accept or return time value specifications. A time value
1998 structure **timespec** specifies a single time value and includes at least the following members:

1999	Member Type	Member Name	Description
2000	time_t	<i>tv_sec</i>	Seconds.
2001	long	<i>tv_nsec</i>	Nanoseconds.
2002			

2003 The *tv_nsec* member is only valid if greater than or equal to zero, and less than the number of
2004 nanoseconds in a second (1 000 million). The time interval described by this structure is (*tv_sec* *
2005 10^9 + *tv_nsec*) nanoseconds.

2006 A time value structure **itimerspec** specifies an initial timer value and a repetition interval for use
2007 by the per-process timer functions. This structure includes at least the following members:

2008	Member Type	Member Name	Description
2009	struct timespec	<i>it_interval</i>	Timer period.
2010	struct timespec	<i>it_value</i>	Timer expiration.
2011			

2012 If the value described by *it_value* is non-zero, it indicates the time to or time of the next timer
2013 expiration (for relative and absolute timer values, respectively). If the value described by *it_value*
2014 is zero, the timer shall be disarmed.

2015 If the value described by *it_interval* is non-zero, it specifies an interval which shall be used in
2016 reloading the timer when it expires; that is, a periodic timer is specified. If the value described by

2017 *it_interval* is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is
2018 specified.

2019 **Timer Event Notification Control Block**

2020 RTS Per-process timers may be created that notify the process of timer expirations by queuing a
2021 realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of
2022 IEEE Std 1003.1-2001, <**signal.h**>, is used in creating such a timer. The **sigevent** structure
2023 contains the signal number and an application-specific data value which shall be used when
2024 notifying the calling process of timer expiration events.

2025 **Manifest Constants**

2026 The following constants are defined in the Base Definitions volume of IEEE Std 1003.1-2001,
2027 <**time.h**>:

2028 **CLOCK_REALTIME** The identifier for the system-wide realtime clock.

2029 **TIMER_ABSTIME** Flag indicating time is absolute with respect to the clock associated
2030 with a timer.

2031 MON **CLOCK_MONOTONIC** The identifier for the system-wide monotonic clock, which is defined
2032 as a clock whose value cannot be set via *clock_settime()* and which
2033 cannot have backward clock jumps. The maximum possible clock
2034 jump is implementation-defined.

2035 MON The maximum allowable resolution for **CLOCK_REALTIME** and **CLOCK_MONOTONIC** clocks
2036 and all time services based on these clocks is represented by **{_POSIX_CLOCKRES_MIN}** and
2037 shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of
2038 resolution for these clocks to provide finer granularity time bases. The actual resolution
2039 supported by an implementation for a specific clock is obtained using the *clock_getres()* function.
2040 If the actual resolution supported for a time service based on one of these clocks differs from the
2041 resolution supported for that clock, the implementation shall document this difference.

2042 MON The minimum allowable maximum value for **CLOCK_REALTIME** and **CLOCK_MONOTONIC**
2043 clocks and all absolute time services based on them is the same as that defined by the ISO C
2044 standard for the **time_t** type. If the maximum value supported by a time service based on one of
2045 these clocks differs from the maximum value supported by that clock, the implementation shall
2046 document this difference.

2047 **Execution Time Monitoring**

2048 CPT If **_POSIX_CPUTIME** is defined, process CPU-time clocks shall be supported in addition to the
2049 clocks described in **Manifest Constants**.

2050 TCT If **_POSIX_THREAD_CPUTIME** is defined, thread CPU-time clocks shall be supported.

2051 CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in the Base Definitions
2052 volume of IEEE Std 1003.1-2001, Section 3.117, CPU Time (Execution Time). The mechanism
2053 used to measure execution time is described in the Base Definitions volume of
2054 IEEE Std 1003.1-2001, Section 4.9, Measurement of Execution Time.

2055 CPT If **_POSIX_CPUTIME** is defined, the following constant of the type **clockid_t** is defined in
2056 <**time.h**>:

2057 **CLOCK_PROCESS_CPUTIME_ID**

2058 When this value of the type **clockid_t** is used in a *clock()* or *timer*()* function call, it is
2059 interpreted as the identifier of the CPU-time clock associated with the process making the

2060 function call.

2061

2062 TCT If `_POSIX_THREAD_CPUTIME` is defined, the following constant of the type **clockid_t** is

2063 defined in **<time.h>**:

2064 **CLOCK_THREAD_CPUTIME_ID**

2065 When this value of the type **clockid_t** is used in a *clock()* or *timer*()* function call, it is

2066 interpreted as the identifier of the CPU-time clock associated with the thread making the

2067 function call.

2068 2.9 Threads

2069 THR The functionality described in this section is dependent on support of the Threads option (and

2070 the rest of this section is not further shaded for this option).

2071 This section defines functionality to support multiple flows of control, called “threads”, within a

2072 process. For the definition of threads, see the Base Definitions volume of IEEE Std 1003.1-2001,

2073 Section 3.393, Thread.

2074 The specific functional areas covered by threads and their scope include:

- 2075 • Thread management: the creation, control, and termination of multiple flows of control in the
- 2076 same process under the assumption of a common shared address space
- 2077 • Synchronization primitives optimized for tightly coupled operation of multiple control flows
- 2078 in a common, shared address space

2079 2.9.1 Thread-Safety

2080 All functions defined by this volume of IEEE Std 1003.1-2001 shall be thread-safe, except that the

2081 following functions¹ need not be thread-safe.

2082 <i>asctime()</i>	<i>ecvt()</i>	<i>gethostent()</i>	<i>getutxline()</i>	<i>putc_unlocked()</i>
2083 <i>basename()</i>	<i>encrypt()</i>	<i>getlogin()</i>	<i>gmtime()</i>	<i>putchar_unlocked()</i>
2084 <i>catgets()</i>	<i>endgrent()</i>	<i>getnetbyaddr()</i>	<i>hcreate()</i>	<i>putenv()</i>
2085 <i>crypt()</i>	<i>endpwent()</i>	<i>getnetbyname()</i>	<i>hdestroy()</i>	<i>pututxline()</i>
2086 <i>ctime()</i>	<i>endutxent()</i>	<i>getnetent()</i>	<i>hsearch()</i>	<i>rand()</i>
2087 <i>dbm_clearerr()</i>	<i>fcvt()</i>	<i>getopt()</i>	<i>inet_ntoa()</i>	<i>readdir()</i>
2088 <i>dbm_close()</i>	<i>ftw()</i>	<i>getprotobyname()</i>	<i>l64a()</i>	<i>setenv()</i>
2089 <i>dbm_delete()</i>	<i>gcvt()</i>	<i>getprotobyname()</i>	<i>lgamma()</i>	<i>setgrent()</i>
2090 <i>dbm_error()</i>	<i>getc_unlocked()</i>	<i>getprotoent()</i>	<i>lgammaf()</i>	<i>setkey()</i>
2091 <i>dbm_fetch()</i>	<i>getchar_unlocked()</i>	<i>getpwent()</i>	<i>lgammal()</i>	<i>setpwent()</i>
2092 <i>dbm_firstkey()</i>	<i>getdate()</i>	<i>getpwnam()</i>	<i>localeconv()</i>	<i>setutxent()</i>
2093 <i>dbm_nextkey()</i>	<i>getenv()</i>	<i>getpwuid()</i>	<i>localtime()</i>	<i>strerror()</i>
2094 <i>dbm_open()</i>	<i>getgrent()</i>	<i>getservbyname()</i>	<i>lrand48()</i>	<i>strtok()</i>
2095 <i>dbm_store()</i>	<i>getgrgid()</i>	<i>getservbyport()</i>	<i>mrnd48()</i>	<i>ttynam()</i>
2096 <i>dirname()</i>	<i>getgrnam()</i>	<i>getservent()</i>	<i>nftw()</i>	<i>unsetenv()</i>

2097 _____

2098 1. The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

2099	<i>derror()</i>	<i>gethostbyaddr()</i>	<i>getutxent()</i>	<i>nl_langinfo()</i>	<i>wcstombs()</i>
2100	<i>drand48()</i>	<i>gethostbyname()</i>	<i>getutxid()</i>	<i>ptsname()</i>	<i>wctomb()</i>

2101 The *ctermid()* and *tmpnam()* functions need not be thread-safe if passed a NULL argument. The
 2102 *wcrtomb()* and *wcsrtombs()* functions need not be thread-safe if passed a NULL *ps* argument.

2103 Implementations shall provide internal synchronization as necessary in order to satisfy this
 2104 requirement.

2105 2.9.2 Thread IDs

2106 Although implementations may have thread IDs that are unique in a system, applications
 2107 should only assume that thread IDs are usable and unique within a single process. The effect of
 2108 calling any of the functions defined in this volume of IEEE Std 1003.1-2001 and passing as an
 2109 argument the thread ID of a thread from another process is unspecified. A conforming
 2110 implementation is free to reuse a thread ID after the thread terminates if it was created with the
 2111 *detachstate* attribute set to *PTHREAD_CREATE_DETACHED* or if *pthread_detach()* or
 2112 *pthread_join()* has been called for that thread. If a thread is detached, its thread ID is invalid for
 2113 use as an argument in a call to *pthread_detach()* or *pthread_join()*.

2114 2.9.3 Thread Mutexes

2115 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same
 2116 processing resources from eventually making forward progress in its execution. Eligibility for
 2117 processing resources is determined by the scheduling policy.

2118 A thread shall become the owner of a mutex, *m*, when one of the following occurs:

- 2119 • It returns successfully from *pthread_mutex_lock()* with *m* as the *mutex* argument.
- 2120 • It returns successfully from *pthread_mutex_trylock()* with *m* as the *mutex* argument.
- 2121 TMO • It returns successfully from *pthread_mutex_timedlock()* with *m* as the *mutex* argument.
- 2122 • It returns (successfully or not) from *pthread_cond_wait()* with *m* as the *mutex* argument
 2123 (except as explicitly indicated otherwise for certain errors).
- 2124 • It returns (successfully or not) from *pthread_cond_timedwait()* with *m* as the *mutex* argument
 2125 (except as explicitly indicated otherwise for certain errors).

2126 The thread shall remain the owner of *m* until one of the following occurs:

- 2127 • It executes *pthread_mutex_unlock()* with *m* as the *mutex* argument
- 2128 • It blocks in a call to *pthread_cond_wait()* with *m* as the *mutex* argument.
- 2129 • It blocks in a call to *pthread_cond_timedwait()* with *m* as the *mutex* argument.

2130 The implementation shall behave as if at all times there is at most one owner of any mutex.

2131 A thread that becomes the owner of a mutex is said to have “acquired” the mutex and the mutex
 2132 is said to have become “locked”; when a thread gives up ownership of a mutex it is said to have
 2133 “released” the mutex and the mutex is said to have become “unlocked”.

2.9.4 Thread Scheduling

The functionality described in this section is dependent on support of the Thread Execution Scheduling option (and the rest of this section is not further shaded for this option).

Thread Scheduling Attributes

In support of the scheduling function, threads have attributes which are accessed through the **pthread_attr_t** thread creation attributes object.

The *contentionscope* attribute defines the scheduling contention scope of the thread to be either PTHREAD_SCOPE_PROCESS or PTHREAD_SCOPE_SYSTEM.

The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling attributes of the creating thread or to have its scheduling values set according to the other scheduling attributes in the **pthread_attr_t** object.

The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute defines the scheduling parameters for the thread. The interaction of threads having different policies within a process is described as part of the definition of those policies.

If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one of the priority-based policies defined under this option, the *schedparam* attribute contains the scheduling priority of the thread. A conforming implementation ensures that the priority value in *schedparam* is in the range associated with the scheduling policy when the thread attributes object is used to create a thread, or when the scheduling attributes of a thread are dynamically modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the *schedparam* attribute supports four new members that are used for the sporadic server scheduling policy. These members are *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The meaning of these attributes is the same as in the definitions that appear under Section 2.8.4 (on page 44).

When a process is created, its single thread has a scheduling policy and associated attributes equal to the process' policy and attributes. The default scheduling contention scope value is implementation-defined. The default values of other scheduling attributes are implementation-defined.

Thread Scheduling Contention Scope

The scheduling contention scope of a thread defines the set of threads with which the thread competes for use of the processing resources. The scheduling operation selects at most one thread to execute on each processor at any point in time and the thread's scheduling attributes (for example, *priority*), whether under process scheduling contention scope or system scheduling contention scope, are the parameters used to determine the scheduling decision.

The scheduling contention scope, in the context of scheduling a mixed scope environment, affects threads as follows:

- A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends for resources with all other threads in the same scheduling allocation domain relative to their system scheduling attributes. The system scheduling attributes of a thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope are the scheduling attributes with which the thread was created. The system scheduling attributes of a thread created with PTHREAD_SCOPE_PROCESS scheduling contention scope are the implementation-defined mapping into system attribute space of the scheduling attributes with which the thread was created.

- Threads created with `PTHREAD_SCOPE_PROCESS` scheduling contention scope contend directly with other threads within their process that were created with `PTHREAD_SCOPE_PROCESS` scheduling contention scope. The contention is resolved based on the threads' scheduling attributes and policies. It is unspecified how such threads are scheduled relative to threads in other processes or threads with `PTHREAD_SCOPE_SYSTEM` scheduling contention scope.

- Conforming implementations shall support the `PTHREAD_SCOPE_PROCESS` scheduling contention scope, the `PTHREAD_SCOPE_SYSTEM` scheduling contention scope, or both.

Scheduling Allocation Domain

Implementations shall support scheduling allocation domains containing one or more processors. It should be noted that the presence of multiple processors does not automatically indicate a scheduling allocation domain size greater than one. Conforming implementations on multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation domains, and could define these scheduling allocation domains on a per-thread, per-process, or per-system basis, depending on the types of applications intended to be supported by the implementation. The scheduling allocation domain is independent of scheduling contention scope, as the scheduling contention scope merely defines the set of threads with which a thread contends for processor resources, while scheduling allocation domain defines the set of processors for which it contends. The semantics of how this contention is resolved among threads for processors is determined by the scheduling policies of the threads.

The choice of scheduling allocation domain size and the level of application control over scheduling allocation domains is implementation-defined. Conforming implementations may change the size of scheduling allocation domains and the binding of threads to scheduling allocation domains at any time.

For application threads with scheduling allocation domains of size equal to one, the scheduling rules defined for `SCHED_FIFO` and `SCHED_RR` shall be used; see **Scheduling Policies** (on page 44). All threads with system scheduling contention scope, regardless of the processes in which they reside, compete for the processor according to their priorities. Threads with process scheduling contention scope compete only with other threads with process scheduling contention scope within their process.

For application threads with scheduling allocation domains of size greater than one, the rules defined for `SCHED_FIFO`, `SCHED_RR`, and `SCHED_SPORADIC` shall be used in an implementation-defined manner. Each thread with system scheduling contention scope competes for the processors in its scheduling allocation domain in an implementation-defined manner according to its priority. Threads with process scheduling contention scope are scheduled relative to other threads within the same scheduling contention scope in the process.

If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the rules defined for `SCHED_SPORADIC` in **Scheduling Policies** (on page 44) shall be used in an implementation-defined manner for application threads whose scheduling allocation domain size is greater than one.

2218 Scheduling Documentation

2219 If `_POSIX_PRIORITY_SCHEDULING` is defined, then any scheduling policies beyond
 2220 TSP `SCHED_OTHER`, `SCHED_FIFO`, `SCHED_RR`, and `SCHED_SPORADIC`, as well as the effects of
 2221 the scheduling policies indicated by these other values, and the attributes required in order to
 2222 support such a policy, are implementation-defined. Furthermore, the implementation shall
 2223 document the effect of all processor scheduling allocation domain values supported for these
 2224 policies.

2225 2.9.5 Thread Cancellation

2226 The thread cancellation mechanism allows a thread to terminate the execution of any other
 2227 thread in the process in a controlled manner. The target thread (that is, the one that is being
 2228 canceled) is allowed to hold cancellation requests pending in a number of ways and to perform
 2229 application-specific cleanup processing when the notice of cancellation is acted upon.

2230 Cancellation is controlled by the cancellation control functions. Each thread maintains its own
 2231 cancelability state. Cancellation may only occur at cancellation points or when the thread is
 2232 asynchronously cancelable.

2233 The thread cancellation mechanism described in this section depends upon programs having set
 2234 *deferred* cancelability state, which is specified as the default. Applications shall also carefully
 2235 follow static lexical scoping rules in their execution behavior. For example, use of *setjmp()*,
 2236 *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary
 2237 scope pop operation results in undefined behavior.

2238 Use of asynchronous cancelability while holding resources which potentially need to be released
 2239 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated
 2240 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

2241 2.9.5.1 Cancelability States

2242 The cancelability state of a thread determines the action taken upon receipt of a cancellation
 2243 request. The thread may control cancellation in a number of ways.

2244 Each thread maintains its own cancelability state, which may be encoded in two bits:

- 2245 1. Cancelability-Enable: When cancelability is `PTHREAD_CANCEL_DISABLE` (as defined in
 2246 the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**), cancellation requests
 2247 against the target thread are held pending. By default, cancelability is set to
 2248 `PTHREAD_CANCEL_ENABLE` (as defined in **<pthread.h>**).
- 2249 2. Cancelability Type: When cancelability is enabled and the cancelability type is
 2250 `PTHREAD_CANCEL_ASYNCHRONOUS` (as defined in **<pthread.h>**), new or pending
 2251 cancellation requests may be acted upon at any time. When cancelability is enabled and
 2252 the cancelability type is `PTHREAD_CANCEL_DEFERRED` (as defined in **<pthread.h>**),
 2253 cancellation requests are held pending until a cancellation point (see below) is reached. If
 2254 cancelability is disabled, the setting of the cancelability type has no immediate effect as all
 2255 cancellation requests are held pending; however, once cancelability is enabled again the
 2256 new type is in effect. The cancelability type is `PTHREAD_CANCEL_DEFERRED` in all
 2257 newly created threads including the thread in which *main()* was first invoked.

2258 2.9.5.2 Cancellation Points

2259 Cancellation points shall occur when a thread is executing the following functions:

2260	<i>accept()</i>	<i>mq_timedsend()</i>	<i>putmsg()</i>	<i>sigpause()</i>
2261	<i>aio_suspend()</i>	<i>msgrcv()</i>	<i>putpmsg()</i>	<i>sigsuspend()</i>
2262	<i>clock_nanosleep()</i>	<i>msgsnd()</i>	<i>pwrite()</i>	<i>sigtimedwait()</i>
2263	<i>close()</i>	<i>msync()</i>	<i>read()</i>	<i>sigwait()</i>
2264	<i>connect()</i>	<i>nanosleep()</i>	<i>readv()</i>	<i>sigwaitinfo()</i>
2265	<i>creat()</i>	<i>open()</i>	<i>recv()</i>	<i>sleep()</i>
2266	<i>fcntl()</i> ²	<i>pause()</i>	<i>recvfrom()</i>	<i>system()</i>
2267	<i>fsync()</i>	<i>poll()</i>	<i>recvmsg()</i>	<i>tcdrain()</i>
2268	<i>getmsg()</i>	<i>pread()</i>	<i>select()</i>	<i>usleep()</i>
2269	<i>getpmsg()</i>	<i>pselect()</i>	<i>sem_timedwait()</i>	<i>wait()</i>
2270	<i>lockf()</i>	<i>pthread_cond_timedwait()</i>	<i>sem_wait()</i>	<i>waitid()</i>
2271	<i>mq_receive()</i>	<i>pthread_cond_wait()</i>	<i>send()</i>	<i>waitpid()</i>
2272	<i>mq_send()</i>	<i>pthread_join()</i>	<i>sendmsg()</i>	<i>write()</i>
2273	<i>mq_timedreceive()</i>	<i>pthread_testcancel()</i>	<i>sendto()</i>	<i>writew()</i>

2274 _____

2275 2. When the *cmd* argument is F_SETLK.

2276 A cancellation point may also occur when a thread is executing the following functions: |

2277	<code>catclose()</code>	<code>ftell()</code>	<code>getwc()</code>	<code>pthread_rwlock_wrlock()</code>
2278	<code>catgets()</code>	<code>ftello()</code>	<code>getwchar()</code>	<code>putc()</code>
2279	<code>catopen()</code>	<code>ftw()</code>	<code>getwd()</code>	<code>putc_unlocked()</code>
2280	<code>closedir()</code>	<code>fwprintf()</code>	<code>glob()</code>	<code>putchar()</code>
2281	<code>closelog()</code>	<code>fwrite()</code>	<code>iconv_close()</code>	<code>putchar_unlocked()</code>
2282	<code>ctermid()</code>	<code>fwscanf()</code>	<code>iconv_open()</code>	<code>puts()</code>
2283	<code>dbm_close()</code>	<code>getc()</code>	<code>ioctl()</code>	<code>pututxline()</code>
2284	<code>dbm_delete()</code>	<code>getc_unlocked()</code>	<code>lseek()</code>	<code>putwc()</code>
2285	<code>dbm_fetch()</code>	<code>getchar()</code>	<code>mkstemp()</code>	<code>putwchar()</code>
2286	<code>dbm_nextkey()</code>	<code>getchar_unlocked()</code>	<code>nftw()</code>	<code>readdir()</code>
2287	<code>dbm_open()</code>	<code>getcwd()</code>	<code>opendir()</code>	<code>readdir_r()</code>
2288	<code>dbm_store()</code>	<code>getdate()</code>	<code>openlog()</code>	<code>remove()</code>
2289	<code>dlclose()</code>	<code>getgrent()</code>	<code>pclose()</code>	<code>rename()</code>
2290	<code>dlopen()</code>	<code>getgrgid()</code>	<code>perror()</code>	<code>rewind()</code>
2291	<code>endgrent()</code>	<code>getgrgid_r()</code>	<code>popen()</code>	<code>rewinddir()</code>
2292	<code>endhostent()</code>	<code>getgrnam()</code>	<code>posix_fadvise()</code>	<code>scanf()</code>
2293	<code>endnetent()</code>	<code>getgrnam_r()</code>	<code>posix_fallocate()</code>	<code>seekdir()</code>
2294	<code>endprotoent()</code>	<code>gethostbyaddr()</code>	<code>posix_madvise()</code>	<code>semop()</code>
2295	<code>endpwent()</code>	<code>gethostbyname()</code>	<code>posix_spawn()</code>	<code>setgrent()</code>
2296	<code>endservent()</code>	<code>gethostent()</code>	<code>posix_spawnnp()</code>	<code>sethostent()</code>
2297	<code>endutxent()</code>	<code>gethostname()</code>	<code>posix_trace_clear()</code>	<code>setnetent()</code>
2298	<code>fclose()</code>	<code>getlogin()</code>	<code>posix_trace_close()</code>	<code>setprotoent()</code>
2299	<code>fcntl()</code> ³	<code>getlogin_r()</code>	<code>posix_trace_create()</code>	<code>setpwent()</code>
2300	<code>fflush()</code>	<code>getnetbyaddr()</code>	<code>posix_trace_create_withlog()</code>	<code>setservent()</code>
2301	<code>fgetc()</code>	<code>getnetbyname()</code>	<code>posix_trace_eventtypelist_getnext_id()</code>	<code>setutxent()</code>
2302	<code>fgetpos()</code>	<code>getnetent()</code>	<code>posix_trace_eventtypelist_rewind()</code>	<code>strerror()</code>
2303	<code>fgets()</code>	<code>getprotobyname()</code>	<code>posix_trace_flush()</code>	<code>syslog()</code>
2304	<code>fgetwc()</code>	<code>getprotobynumber()</code>	<code>posix_trace_get_attr()</code>	<code>tmpfile()</code>
2305	<code>fgetws()</code>	<code>getprotoent()</code>	<code>posix_trace_get_filter()</code>	<code>tmpnam()</code>
2306	<code>fopen()</code>	<code>getpwent()</code>	<code>posix_trace_get_status()</code>	<code>ttyname()</code>
2307	<code>fprintf()</code>	<code>getpwnam()</code>	<code>posix_trace_getnext_event()</code>	<code>ttyname_r()</code>
2308	<code>fputc()</code>	<code>getpwnam_r()</code>	<code>posix_trace_open()</code>	<code>ungetc()</code>
2309	<code>fputs()</code>	<code>getpwuid()</code>	<code>posix_trace_rewind()</code>	<code>ungetwc()</code>
2310	<code>fputwc()</code>	<code>getpwuid_r()</code>	<code>posix_trace_set_filter()</code>	<code>unlink()</code>
2311	<code>fputws()</code>	<code>gets()</code>	<code>posix_trace_shutdown()</code>	<code>vfprintf()</code>
2312	<code>fread()</code>	<code>getservbyname()</code>	<code>posix_trace_timedgetnext_event()</code>	<code>vfwprintf()</code>
2313	<code>freopen()</code>	<code>getservbyport()</code>	<code>posix_typed_mem_open()</code>	<code>vprintf()</code>
2314	<code>fscanf()</code>	<code>getservent()</code>	<code>printf()</code>	<code>vwprintf()</code>
2315	<code>fseek()</code>	<code>getutxent()</code>	<code>pthread_rwlock_rdlock()</code>	<code>wprintf()</code>
2316	<code>fseeko()</code>	<code>getutxid()</code>	<code>pthread_rwlock_timedrdlock()</code>	<code>wscanf()</code>
2317	<code>fsetpos()</code>	<code>getutxline()</code>	<code>pthread_rwlock_timedwrlock()</code>	

2318 An implementation shall not introduce cancellation points into any other functions specified in |
 2319 this volume of IEEE Std 1003.1-2001.

2320 _____

2321 3. For any value of the *cmd* argument.

2322 The side effects of acting upon a cancellation request while suspended during a call of a function
 2323 are the same as the side effects that may be seen in a single-threaded program when a call to a
 2324 function is interrupted by a signal and the given function returns [EINTR]. Any such side effects
 2325 occur before any cancellation cleanup handlers are called.

2326 Whenever a thread has cancelability enabled and a cancellation request has been made with that
 2327 thread as the target, and the thread then calls any function that is a cancellation point (such as
 2328 *pthread_testcancel()* or *read()*), the cancellation request shall be acted upon before the function
 2329 returns. If a thread has cancelability enabled and a cancellation request is made with the thread
 2330 as a target while the thread is suspended at a cancellation point, the thread shall be awakened
 2331 and the cancellation request shall be acted upon. However, if the thread is suspended at a
 2332 cancellation point and the event for which it is waiting occurs before the cancellation request is
 2333 acted upon, it is unspecified whether the cancellation request is acted upon or whether the
 2334 cancellation request remains pending and the thread resumes normal execution.

2335 2.9.5.3 Thread Cancellation Cleanup Handlers

2336 Each thread maintains a list of cancellation cleanup handlers. The programmer uses the
 2337 *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions to place routines on and remove
 2338 routines from this list.

2339 When a cancellation request is acted upon, the routines in the list are invoked one by one in
 2340 LIFO sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked
 2341 (First Out). The thread invokes the cancellation cleanup handler with cancellation disabled until
 2342 the last cancellation cleanup handler returns. When the cancellation cleanup handler for a scope
 2343 is invoked, the storage for that scope remains valid. If the last cancellation cleanup handler
 2344 returns, thread execution is terminated and a status of PTHREAD_CANCELED is made
 2345 available to any threads joining with the target. The symbolic constant PTHREAD_CANCELED
 2346 expands to a constant expression of type (**void ***) whose value matches no pointer to an object in
 2347 memory nor the value NULL.

2348 The cancellation cleanup handlers are also invoked when the thread calls *pthread_exit()*.

2349 A side effect of acting upon a cancellation request while in a condition variable wait is that the
 2350 mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread
 2351 is no longer considered to be waiting for the condition and the thread shall not have consumed
 2352 any pending condition signals on the condition.

2353 A cancellation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

2354 2.9.5.4 Async-Cancel Safety

2355 The *pthread_cancel()*, *pthread_setcancelstate()*, and *pthread_setcanceltype()* functions are defined to
 2356 be async-cancel safe.

2357 No other functions in this volume of IEEE Std 1003.1-2001 are required to be async-cancel-safe.

2358 2.9.6 Thread Read-Write Locks

2359 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
 2360 read-only access to data while allowing only one thread to have exclusive write access at any
 2361 given time. They are typically used to protect data that is read more frequently than it is
 2362 changed.

2363 One or more readers acquire read access to the resource by performing a read lock operation on
 2364 the associated read-write lock. A writer acquires exclusive write access by performing a write
 2365 lock operation. Basically, all readers exclude any writers and a writer excludes all readers and
 2366 any other writers.

2367 A thread that has blocked on a read-write lock (for example, has not yet returned from a
 2368 *pthread_rwlock_rdlock()* or *pthread_rwlock_wrlock()* call) shall not prevent any unblocked thread
 2369 that is eligible to use the same processing resources from eventually making forward progress in
 2370 its execution. Eligibility for processing resources shall be determined by the scheduling policy.

2371 Read-write locks can be used to synchronize threads in the current process and other processes if
 2372 they are allocated in memory that is writable and shared among the cooperating processes and
 2373 have been initialized for this behavior.

2374 2.9.7 Thread Interactions with Regular File Operations

2375 All of the functions *chmod()*, *close()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lseek()*, *open()*, *read()*,
 2376 *readlink()*, *stat()*, *symlink()*, and *write()* shall be atomic with respect to each other in the effects
 2377 specified in IEEE Std 1003.1-2001 when they operate on regular files. If two threads each call one
 2378 of these functions, each call shall either see all of the specified effects of the other call, or none of
 2379 them.

2380 2.10 Sockets

2381 A socket is an endpoint for communication using the facilities described in this section. A socket
 2382 is created with a specific socket type, described in Section 2.10.6 (on page 59), and is associated
 2383 with a specific protocol, detailed in Section 2.10.3 (on page 59). A socket is accessed via a file
 2384 descriptor obtained when the socket is created.

2385 2.10.1 Address Families

2386 All network protocols are associated with a specific address family. An address family provides
 2387 basic services to the protocol implementation to allow it to function within a specific network
 2388 environment. These services may include packet fragmentation and reassembly, routing,
 2389 addressing, and basic transport. An address family is normally comprised of a number of
 2390 protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not
 2391 required that an address family support all socket types. An address family may contain
 2392 multiple protocols supporting the same socket abstraction.

2393 Section 2.10.17 (on page 66), Section 2.10.19 (on page 67), and Section 2.10.20 (on page 67),
 2394 respectively, describe the use of sockets for local UNIX connections, for Internet protocols based
 2395 on IPv4, and for Internet protocols based on IPv6.

2396 2.10.2 Addressing

2397 An address family defines the format of a socket address. All network addresses are described
 2398 using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of
 2399 IEEE Std 1003.1-2001, <sys/socket.h>. However, each address family imposes finer and more
 2400 specific structure, generally defining a structure with fields specific to the address family. The
 2401 field *sa_family* in the **sockaddr** structure contains the address family identifier, specifying the
 2402 format of the *sa_data* area. The size of the *sa_data* area is unspecified.

2403 2.10.3 Protocols

2404 A protocol supports one of the socket abstractions detailed in Section 2.10.6. Selecting a protocol
 2405 involves specifying the address family, socket type, and protocol number to the *socket()*
 2406 function. Certain semantics of the basic socket abstractions are protocol-specific. All protocols
 2407 are expected to support the basic model for their particular socket type, but may, in addition,
 2408 provide non-standard facilities or extensions to a mechanism.

2409 2.10.4 Routing

2410 Sockets provides packet routing facilities. A routing information database is maintained, which
 2411 is used in selecting the appropriate network interface when transmitting packets.

2412 2.10.5 Interfaces

2413 Each network interface in a system corresponds to a path through which messages can be sent
 2414 and received. A network interface usually has a hardware device associated with it, though
 2415 certain interfaces such as the loopback interface, do not.

2416 2.10.6 Socket Types

2417 A socket is created with a specific type, which defines the communication semantics and which
 2418 RS allows the selection of an appropriate communication protocol. Four types are defined:
 2419 **SOCK_RAW**, **SOCK_STREAM**, **SOCK_SEQPACKET**, and **SOCK_DGRAM**. Implementations
 2420 may specify additional socket types.

2421 The **SOCK_STREAM** socket type provides reliable, sequenced, full-duplex octet streams
 2422 between the socket and a peer to which the socket is connected. A socket of type
 2423 **SOCK_STREAM** must be in a connected state before any data may be sent or received. Record
 2424 boundaries are not maintained; data sent on a stream socket using output operations of one size
 2425 may be received using input operations of smaller or larger sizes without loss of data. Data may
 2426 be buffered; successful return from an output function does not imply that the data has been
 2427 delivered to the peer or even transmitted from the local system. If data cannot be successfully
 2428 transmitted within a given time then the connection is considered broken, and subsequent
 2429 operations shall fail. A SIGPIPE signal is raised if a thread sends on a broken stream (one that is
 2430 no longer connected). Support for an out-of-band data transmission facility is protocol-specific.

2431 The **SOCK_SEQPACKET** socket type is similar to the **SOCK_STREAM** type, and is also
 2432 connection-oriented. The only difference between these types is that record boundaries are
 2433 maintained using the **SOCK_SEQPACKET** type. A record can be sent using one or more output
 2434 operations and received using one or more input operations, but a single operation never
 2435 transfers parts of more than one record. Record boundaries are visible to the receiver via the
 2436 **MSG_EOR** flag in the received message flags returned by the *recvmsg()* function. It is protocol-
 2437 specific whether a maximum record size is imposed.

2438 The **SOCK_DGRAM** socket type supports connectionless data transfer which is not necessarily
 2439 acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or

2440 broadcast) in each output operation, and incoming datagrams may be received from multiple
 2441 sources. The source address of each datagram is available when receiving the datagram. An
 2442 application may also pre-specify a peer address, in which case calls to output functions shall
 2443 send to the pre-specified peer. If a peer has been specified, only datagrams from that peer shall
 2444 be received. A datagram must be sent in a single output operation, and must be received in a
 2445 single input operation. The maximum size of a datagram is protocol-specific; with some
 2446 protocols, the limit is implementation-defined. Output datagrams may be buffered within the
 2447 system; thus, a successful return from an output function does not guarantee that a datagram is
 2448 actually sent or received. However, implementations should attempt to detect any errors
 2449 possible before the return of an output function, reporting any error by an unsuccessful return
 2450 value.

2451 RS The SOCK_RAW socket type is similar to the SOCK_DGRAM type. It differs in that it is
 2452 normally used with communication providers that underlie those used for the other socket
 2453 types. For this reason, the creation of a socket with type SOCK_RAW shall require appropriate
 2454 privilege. The format of datagrams sent and received with this socket type generally include
 2455 specific protocol headers, and the formats are protocol-specific and implementation-defined.

2456 **2.10.7 Socket I/O Mode**

2457 The I/O mode of a socket is described by the O_NONBLOCK file status flag which pertains to
 2458 the open file description for the socket. This flag is initially off when a socket is created, but may
 2459 be set and cleared by the use of the F_SETFL command of the *fcntl()* function.

2460 When the O_NONBLOCK flag is set, functions that would normally block until they are
 2461 complete shall either return immediately with an error, or shall complete asynchronously to the
 2462 execution of the calling process. Data transfer operations (the *read()*, *write()*, *send()*, and *recv()*
 2463 functions) shall complete immediately, transfer only as much as is available, and then return
 2464 without blocking, or return an error indicating that no transfer could be made without blocking.
 2465 The *connect()* function initiates a connection and shall return without blocking when
 2466 O_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection
 2467 was initiated successfully, but that it has not yet completed.

2468 **2.10.8 Socket Owner**

2469 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or
 2470 process group ID using the F_SETOWN command of the *fcntl()* function.

2471 **2.10.9 Socket Queue Limits**

2472 The transmit and receive queue sizes for a socket are set when the socket is created. The default
 2473 sizes used are both protocol-specific and implementation-defined. The sizes may be changed
 2474 using the *setsockopt()* function.

2475 **2.10.10 Pending Error**

2476 Errors may occur asynchronously, and be reported to the socket in response to input from the
 2477 network protocol. The socket stores the pending error to be reported to a user of the socket at the
 2478 next opportunity. The error is returned in response to a subsequent *send()*, *recv()*, or *getsockopt()*
 2479 operation on the socket, and the pending error is then cleared.

2480 2.10.11 Socket Receive Queue

2481 A socket has a receive queue that buffers data when it is received by the system until it is
 2482 removed by a receive call. Depending on the type of the socket and the communication provider,
 2483 the receive queue may also contain ancillary data such as the addressing and other protocol data
 2484 associated with the normal data in the queue, and may contain out-of-band or expedited data.
 2485 The limit on the queue size includes any normal, out-of-band data, datagram source addresses,
 2486 and ancillary data in the queue. The description in this section applies to all sockets, even though
 2487 some elements cannot be present in some instances.

2488 The contents of a receive buffer are logically structured as a series of data segments with
 2489 associated ancillary data and other information. A data segment may contain normal data or
 2490 out-of-band data, but never both. A data segment may complete a record if the protocol
 2491 supports records (always true for types SOCK_SEQPACKET and SOCK_DGRAM). A record
 2492 may be stored as more than one segment; the complete record might never be present in the
 2493 receive buffer at one time, as a portion might already have been returned to the application, and
 2494 another portion might not yet have been received from the communications provider. A data
 2495 segment may contain ancillary protocol data, which is logically associated with the segment.
 2496 Ancillary data is received as if it were queued along with the first normal data octet in the
 2497 segment (if any). A segment may contain ancillary data only, with no normal or out-of-band
 2498 data. For the purposes of this section, a datagram is considered to be a data segment that
 2499 terminates a record, and that includes a source address as a special type of ancillary data. Data
 2500 segments are placed into the queue as data is delivered to the socket by the protocol. Normal
 2501 data segments are placed at the end of the queue as they are delivered. If a new segment
 2502 contains the same type of data as the preceding segment and includes no ancillary data, and if
 2503 the preceding segment does not terminate a record, the segments are logically merged into a
 2504 single segment.

2505 The receive queue is logically terminated if an end-of-file indication has been received or a
 2506 connection has been terminated. A segment shall be considered to be terminated if another
 2507 segment follows it in the queue, if the segment completes a record, or if an end-of-file or other
 2508 connection termination has been reported. The last segment in the receive queue shall also be
 2509 considered to be terminated while the socket has a pending error to be reported.

2510 A receive operation shall never return data or ancillary data from more than one segment.

2511 2.10.12 Socket Out-of-Band Data State

2512 The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed
 2513 in the socket receive queue, either at the end of the queue or before all normal data in the queue.
 2514 In this case, out-of-band data is returned to an application program by a normal receive call.
 2515 Out-of-band data may also be queued separately rather than being placed in the socket receive
 2516 queue, in which case it shall be returned only in response to a receive call that requests out-of-
 2517 band data. It is protocol-specific whether an out-of-band data mark is placed in the receive
 2518 queue to demarcate data preceding the out-of-band data and following the out-of-band data. An
 2519 out-of-band data mark is logically an empty data segment that cannot be merged with other
 2520 segments in the queue. An out-of-band data mark is never returned in response to an input
 2521 operation. The *socketmark()* function can be used to test whether an out-of-band data mark is the
 2522 first element in the queue. If an out-of-band data mark is the first element in the queue when an
 2523 input function is called without the MSG_PEEK option, the mark is removed from the queue and
 2524 the following data (if any) is processed as if the mark had not been present.

2.10.13 Connection Indication Queue

Sockets that are used to accept incoming connections maintain a queue of outstanding connection indications. This queue is a list of connections that are awaiting acceptance by the application; see *listen()*.

2.10.14 Signals

One category of event at the socket interface is the generation of signals. These signals report protocol events or process errors relating to the state of the socket. The generation or delivery of a signal does not change the state of the socket, although the generation of the signal may have been caused by a state change.

The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no longer able to send. In addition, the send operation fails with the error [EPIPE].

If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the status of the socket is specified in Section 2.10.17 (on page 66), Section 2.10.19 (on page 67), and Section 2.10.20 (on page 67). Depending on the protocol, the expedited data may or may not have arrived at the time of signal generation.

2.10.15 Asynchronous Errors

If any of the following conditions occur asynchronously for a socket, the corresponding value listed below shall become the pending error for the socket:

[ECONNABORTED]

The connection was aborted locally.

[ECONNREFUSED]

For a connection-mode socket attempting a non-blocking connection, the attempt to connect was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram was forcefully rejected.

[ECONNRESET]

The peer has aborted the connection.

[EHOSTDOWN]

The destination host has been determined to be down or disconnected.

[EHOSTUNREACH]

The destination host is not reachable.

[EMSGSIZE]

For a connectionless-mode socket, the size of a previously sent datagram prevented delivery.

[ENETDOWN]

The local network connection is not operational.

[ENETRESET]

The connection was aborted by the network.

[ENETUNREACH]

The destination network is not reachable.

2.10.16 Use of Options

There are a number of socket options which either specialize the behavior of a socket or provide useful information. These options may be set at different protocol levels and are always present at the uppermost “socket” level.

Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions allow an application program to customize the behavior and characteristics of a socket to provide the desired effect.

All of the options have default values. The type and meaning of these values is defined by the protocol level to which they apply. Instead of using the default values, an application program may choose to customize one or more of the options. However, in the bulk of cases, the default values are sufficient for the application.

Some of the options are used to enable or disable certain behavior within the protocol modules (for example, turn on debugging) while others may be used to set protocol-specific information (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is introduced, its effect on the underlying protocol modules is described.

Table 2-1 shows the value for the socket level.

Table 2-1 Value of Level for Socket Options

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

Table 2-2 (on page 64) lists those options present at the socket level; that is, when the *level* parameter of the *getsockopt()* or *setsockopt()* function is SOL_SOCKET, the types of the option value parameters associated with each option, and a brief synopsis of the meaning of the option value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with *setsockopt()* on all types of socket.

Table 2-2 Socket-Level Options

Option	Parameter Type	Parameter Meaning
SO_BROADCAST	int	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	int	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	int	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	int	Requests and clears pending error information on the socket (<i>getsockopt()</i> only).
SO_KEEPALIVE	int	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	struct linger	Specify actions to be taken for queued, unsent data on <i>close()</i> : linger on/off and linger time in seconds.
SO_OOBINLINE	int	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	int	Size of receive buffer (in bytes).
SO_RCVLOWAT	int	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	struct timeval	Timeout value for a socket receive operation.
SO_REUSEADDR	int	Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).
SO_SNDBUF	int	Size of send buffer (in bytes).
SO_SNDLOWAT	int	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	struct timeval	Timeout value for a socket send operation.
SO_TYPE	int	Identify socket type (<i>getsockopt()</i> only).

The SO_BROADCAST option requests permission to send broadcast datagrams on the socket. Support for SO_BROADCAST is protocol-specific. The default for SO_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled.

The SO_DEBUG option enables debugging in the underlying protocol modules. This can be useful for tracing the behavior of the underlying protocol modules during normal system operation. The semantics of the debug reports are implementation-defined. The default value for SO_DEBUG is for debugging to be turned off.

The SO_DONTROUTE option requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. It is protocol-specific whether this option has any effect and how the outgoing network interface is chosen. Support for this option with each protocol is implementation-defined.

The SO_ERROR option is used only on *getsockopt()*. When this option is specified, *getsockopt()* shall return any pending error on the socket and clear the error status. It shall return a value of 0 if there is no pending error. SO_ERROR may be used to check for asynchronous errors on connected connectionless-mode sockets or for other types of asynchronous errors. SO_ERROR has no default value.

2636 The SO_KEEPAALIVE option enables the periodic transmission of messages on a connected
2637 socket. The behavior of this option is protocol-specific. The default value for SO_KEEPAALIVE is
2638 zero, specifying that this capability is turned off.

2639 The SO_LINGER option controls the action of the interface when unsent messages are queued
2640 on a socket and a *close()* is performed. The details of this option are protocol-specific. The
2641 default value for SO_LINGER is zero, or off, for the *l_onoff* element of the option value and zero
2642 seconds for the linger time specified by the *l_linger* element.

2643 The SO_OOINLINE option is valid only on protocols that support out-of-band data. The
2644 SO_OOINLINE option requests that out-of-band data be placed in the normal data input queue
2645 as received; it is then accessible using the *read()* or *recv()* functions without the MSG_OOB flag
2646 set. The default for SO_OOINLINE is off; that is, for out-of-band data not to be placed in the
2647 normal data input queue.

2648 The SO_RCVBUF option requests that the buffer space allocated for receive operations on this
2649 socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer
2650 size for high volume connections, or may decrease buffer size to limit the possible backlog of
2651 incoming data. The default value for the SO_RCVBUF option value is implementation-defined,
2652 and may vary by protocol.

2653 The maximum value for the option for a socket may be obtained by the use of the *fpathconf()*
2654 function, using the value `_PC_SOCK_MAXBUF`.

2655 The SO_RCVLOWAT option sets the minimum number of bytes to process for socket input
2656 operations. In general, receive calls block until any (non-zero) amount of data is received, then
2657 return the smaller of the amount available or the amount requested. The default value for
2658 SO_RCVLOWAT is 1, and does not affect the general case. If SO_RCVLOWAT is set to a larger
2659 value, blocking receive calls normally wait until they have received the smaller of the low water
2660 mark value or the requested amount. Receive calls may still return less than the low water mark
2661 if an error occurs, a signal is caught, or the type of data next in the receive queue is different
2662 from that returned (for example, out-of-band data). As mentioned previously, the default value
2663 for SO_RCVLOWAT is 1 byte. It is implementation-defined whether the SO_RCVLOWAT option
2664 can be set.

2665 The SO_RCVTIMEO option is an option to set a timeout value for input operations. It accepts a
2666 **timeval** structure with the number of seconds and microseconds specifying the limit on how
2667 long to wait for an input operation to complete. If a receive operation has blocked for this much
2668 time without receiving additional data, it shall return with a partial count or *errno* shall be set to
2669 [EWOULDBLOCK] if no data were received. The default for this option is the value zero, which
2670 indicates that a receive operation will not time out. It is implementation-defined whether the
2671 SO_RCVTIMEO option can be set.

2672 The SO_REUSEADDR option indicates that the rules used in validating addresses supplied in a
2673 *bind()* should allow reuse of local addresses. Operation of this option is protocol-specific. The
2674 default value for SO_REUSEADDR is off; that is, reuse of local addresses is not permitted.

2675 The SO_SNDBUF option requests that the buffer space allocated for send operations on this
2676 socket be set to the value, in bytes, of the option value. The default value for the SO_SNDBUF
2677 option value is implementation-defined, and may vary by protocol. The maximum value for the
2678 option for a socket may be obtained by the use of the *fpathconf()* function, using the value
2679 `_PC_SOCK_MAXBUF`.

2680 The SO_SNDLOWAT option sets the minimum number of bytes to process for socket output
2681 operations. Most output operations process all of the data supplied by the call, delivering data to
2682 the protocol for transmission and blocking as necessary for flow control. Non-blocking output
2683 operations process as much data as permitted subject to flow control without blocking, but

process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. A *select()* operation testing the ability to write to a socket shall return true only if the send low water mark could be processed. The default value for `SO_SNDLOWAT` is implementation-defined and protocol-specific. It is implementation-defined whether the `SO_SNDLOWAT` option can be set.

The `SO_SNDTIMEO` option is an option to set a timeout value for the amount of time that an output function shall block because flow control prevents data from being sent. As noted in Table 2-2 (on page 64), the option value is a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an output operation to complete. If a send operation has blocked for this much time, it shall return with a partial count or *errno* set to `[EWOULDBLOCK]` if no data were sent. The default for this option is the value zero, which indicates that a send operation will not time out. It is implementation-defined whether the `SO_SNDTIMEO` option can be set.

The `SO_TYPE` option is used only on *getsockopt()*. When this option is specified, *getsockopt()* shall return the type of the socket (for example, `SOCK_STREAM`). This option is useful to servers that inherit sockets on start-up. `SO_TYPE` has no default value.

2.10.17 Use of Sockets for Local UNIX Connections

Support for UNIX domain sockets is mandatory.

UNIX domain sockets provide process-to-process communication in a single system.

2.10.17.1 Headers

The symbolic constant `AF_UNIX` defined in the `<sys/socket.h>` header is used to identify the UNIX domain address family. The `<sys/un.h>` header contains other definitions used in connection with UNIX domain sockets. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

The **sockaddr_storage** structure defined in `<sys/socket.h>` shall be large enough to accommodate a **sockaddr_un** structure (see the `<sys/un.h>` header defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_un** structures and used to access the fields of those structures without alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_un** structure, the *ss_family* field maps onto the *sun_family* field.

2.10.18 Use of Sockets over Internet Protocols

When a socket is created in the Internet family with a protocol value of zero, the implementation shall use the protocol listed below for the type of socket created.

`SOCK_STREAM` `IPPROTO_TCP`.

`SOCK_DGRAM` `IPPROTO_UDP`.

RS `SOCK_RAW` `IPPROTO_RAW`.

`SOCK_SEQPACKET` Unspecified.

RS A raw interface to IP is available by creating an Internet socket of type `SOCK_RAW`. The default protocol for type `SOCK_RAW` shall be identified in the IP header with the value `IPPROTO_RAW`. Applications should not use the default protocol when creating a socket with type `SOCK_RAW`, but should identify a specific protocol by value. The ICMP control protocol is accessible from a raw socket by specifying a value of `IPPROTO_ICMP` for protocol.

2.10.19 Use of Sockets over Internet Protocols Based on IPv4

Support for sockets over Internet protocols based on IPv4 is mandatory.

2.10.19.1 Headers

The symbolic constant `AF_INET` defined in the `<sys/socket.h>` header is used to identify the IPv4 Internet address family. The `<netinet/in.h>` header contains other definitions used in connection with IPv4 Internet sockets. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to accommodate a `sockaddr_in` structure (see the `<netinet/in.h>` header defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an appropriate boundary so that pointers to it can be cast as pointers to `sockaddr_in` structures and used to access the fields of those structures without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_in` structure, the `ss_family` field maps onto the `sin_family` field.

2.10.20 Use of Sockets over Internet Protocols Based on IPv6

This section describes extensions to support sockets over Internet protocols based on IPv6. This functionality is dependent on support of the IPV6 option (and the rest of this section is not further shaded for this option).

To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain circumstances, also be used in connection with IPv4; see Section 2.10.20.2 (on page 68).

2.10.20.1 Addressing

IPv6 overcomes the addressing limitations of previous versions by using 128-bit addresses instead of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.

There are three kinds of IPv6 address:

Unicast

Identifies a single interface.

A unicast address can be global, link-local (designed for use on a single link), or site-local (designed for systems not connected to the Internet). Link-local and site-local addresses need not be globally unique.

Anycast

Identifies a set of interfaces such that a packet sent to the address can be delivered to any member of the set.

An anycast address is similar to a unicast address; the nodes to which an anycast address is assigned must be explicitly configured to know that it is an anycast address.

Multicast

Identifies a set of interfaces such that a packet sent to the address should be delivered to every member of the set.

An application can send multicast datagrams by simply specifying an IPv6 multicast address in the `address` argument of `sendto()`. To receive multicast datagrams, an application must join the multicast group (using `setsockopt()` with `IPV6_JOIN_GROUP`) and must bind to the socket the UDP port on which datagrams will be received. Some applications should also bind the multicast group address to the socket, to prevent other datagrams destined to that port from being delivered to the socket.

2770 A multicast address can be global, node-local, link-local, site-local, or organization-local.

2771 The following special IPv6 addresses are defined:

2772 Unspecified

2773 An address that is not assigned to any interface and is used to indicate the absence of an

2774 address.

2775 Loopback

2776 A unicast address that is not assigned to any interface and can be used by a node to send

2777 packets to itself.

2778 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:

2779 IPv4-compatible addresses

2780 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”

2781 through IPv4.

2782 IPv4-mapped addresses

2783 These are used to represent IPv4 addresses in IPv6 address format; see Section 2.10.20.2.

2784 Note that the unspecified address and the loopback address must not be treated as IPv4-

2785 compatible addresses.

2786 2.10.20.2 Compatibility with IPv4

2787 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,

2788 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the

2789 *getaddrinfo()* function when the specified host has only IPv4 addresses.

2790 Applications can use AF_INET6 sockets to open TCP connections to IPv4 nodes, or send UDP

2791 packets to IPv4 nodes, by simply encoding the destination’s IPv4 address as an IPv4-mapped

2792 IPv6 address, and passing that address, within a **sockaddr_in6** structure, in the *connect()*,

2793 *sendto()*, or *sendmsg()* function. When applications use AF_INET6 sockets to accept TCP

2794 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return

2795 the peer’s address to the application in the *accept()*, *recvfrom()*, *recvmsg()*, or *getpeername()*

2796 function using a **sockaddr_in6** structure encoded this way. If a node has an IPv4 address, then

2797 the implementation shall allow applications to communicate using that address via an

2798 AF_INET6 socket. In such a case, the address will be represented at the API by the

2799 corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an AF_INET6

2800 socket bound to **in6addr_any** to receive inbound connections and packets destined to one of the

2801 node’s IPv4 addresses.

2802 An application can use AF_INET6 sockets to bind to a node’s IPv4 address by specifying the

2803 address as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure in the *bind()* function. For

2804 an AF_INET6 socket bound to a node’s IPv4 address, the system shall return the address in the

2805 *getsockname()* function as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure.

2806 2.10.20.3 Interface Identification

2807 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;

2808 zero is not used. There may be gaps so that there is no current interface for a particular positive

2809 index. Each interface also has a unique implementation-defined name.

2810 2.10.20.4 Options

2811 The following options apply at the IPPROTO_IPV6 level:

2812 IPV6_JOIN_GROUP

2813 When set via *setsockopt()*, it joins the application to a multicast group on an interface
 2814 (identified by its index) and addressed by a given multicast address, enabling packets sent
 2815 to that address to be read via the socket. If the interface index is specified as zero, the
 2816 system selects the interface (for example, by looking up the address in a routing table and
 2817 using the resulting interface).

2818 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

2819 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

2820 IPV6_LEAVE_GROUP

2821 When set via *setsockopt()*, it removes the application from the multicast group on an
 2822 interface (identified by its index) and addressed by a given multicast address.

2823 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

2824 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

2825 IPV6_MULTICAST_HOPS

2826 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the
 2827 socket. Its possible values are the same as those of IPV6_UNICAST_HOPS. If the
 2828 IPV6_MULTICAST_HOPS option is not set, a value of 1 is assumed. This option can be set
 2829 via *setsockopt()* and read via *getsockopt()*.

2830 The parameter type of this option is a pointer to an **int**. (Default value: 1)

2831 IPV6_MULTICAST_IF

2832 The index of the interface to be used for outgoing multicast packets. It can be set via
 2833 *setsockopt()* and read via *getsockopt()*. If the interface index is specified as zero, the system
 2834 selects the interface (for example, by looking up the address in a routing table and using the
 2835 resulting interface).

2836 The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)

2837 IPV6_MULTICAST_LOOP

2838 This option controls whether outgoing multicast packets should be delivered back to the
 2839 local application when the sending interface is itself a member of the destination multicast
 2840 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an
 2841 [EINVAL] error. This option can be set via *setsockopt()* and read via *getsockopt()*.

2842 The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean
 2843 value. (Default value: 1)

2844 IPV6_UNICAST_HOPS

2845 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the
 2846 socket. If the option is not set, or is set to -1, the system selects a default value. Attempts to
 2847 set a value less than -1 or greater than 255 shall result in an [EINVAL] error. This option can
 2848 be set via *setsockopt()* and read via *getsockopt()*.

2849 The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

2850 IPV6_V6ONLY

2851 This socket option restricts AF_INET6 sockets to IPv6 communications only. AF_INET6
 2852 sockets may be used for both IPv4 and IPv6 communications. Some applications may want
 2853 to restrict their use of an AF_INET6 socket to IPv6 communications only. For these

2854 applications, the IPv6_V6ONLY socket option is defined. When this option is turned on, the
 2855 socket can be used to send and receive IPv6 packets only. This is an IPPROTO_IPV6-level
 2856 option.

2857 The parameter type of this option is a pointer to an **int** which is used as a Boolean value.
 2858 (Default value: 0)

2859 An [EOPNOTSUPP] error shall result if IPV6_JOIN_GROUP or IPV6_LEAVE_GROUP is used
 2860 with *getsockopt()*.

2861 2.10.20.5 Headers

2862 The symbolic constant AF_INET6 is defined in the **<sys/socket.h>** header to identify the IPv6
 2863 Internet address family. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13,
 2864 Headers.

2865 The **sockaddr_storage** structure defined in **<sys/socket.h>** shall be large enough to
 2866 accommodate a **sockaddr_in6** structure (see the **<netinet/in.h>** header defined in the Base
 2867 Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an
 2868 appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_in6** structures
 2869 and used to access the fields of those structures without alignment problems. When a
 2870 **sockaddr_storage** structure is cast as a **sockaddr_in6** structure, the *ss_family* field maps onto the
 2871 *sin6_family* field.

2872 The **<netinet/in.h>**, **<arpa/inet.h>**, and **<netdb.h>** headers contain other definitions used in
 2873 connection with IPv6 Internet sockets; see the Base Definitions volume of IEEE Std 1003.1-2001,
 2874 Chapter 13, Headers.

2875 2.11 Tracing

2876 TRC This section describes extensions to support tracing of user applications. This functionality is
 2877 dependent on support of the Trace option (and the rest of this section is not further shaded for
 2878 this option).

2879 The tracing facilities defined in IEEE Std 1003.1-2001 allow a process to select a set of trace event
 2880 types, to activate a trace stream of the selected trace events as they occur in the flow of
 2881 execution, and to retrieve the recorded trace events.

2882 The tracing operation relies on three logically different components: the traced process, the
 2883 controller process, and the analyzer process. During the execution of the traced process, when a
 2884 trace point is reached, a trace event is recorded into the trace streams created for that process in
 2885 which the associated trace event type identifier is not being filtered out. The controller process
 2886 controls the operation of recording the trace events into the trace stream. It shall be able to:

- 2887 • Initialize the attributes of a trace stream
- 2888 • Create the trace stream (for a specified traced process) using those attributes
- 2889 • Start and stop tracing for the trace stream
- 2890 • Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- 2891 • Shut a trace stream down

2892 These operations can be done for an active trace stream. The analyzer process retrieves the
 2893 traced events either at runtime, when the trace stream has not yet been shut down, but is still
 2894 recording trace events; or after opening a trace log that had been previously recorded and shut
 2895 down. These three logically different operations can be performed by the same process, or can be

distributed into different processes.

A trace stream identifier can be created by a call to *posix_trace_create()*, *posix_trace_create_withlog()*, or *posix_trace_open()*. The *posix_trace_create()* and *posix_trace_create_withlog()* functions should be used by a controller process. The *posix_trace_open()* should be used by an analyzer process.

The tracing functions can serve different purposes. One purpose is debugging the possibly pre-instrumented code, while another is post-mortem fault analysis. These two potential uses differ in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and permits focusing on expected information; while the second needs comprehensive trace capabilities in order to be able to record all types of information.

The events to be traced belong to two classes:

1. User trace events (generated by the application instrumentation)
2. System trace events (generated by the operating system)

The trace interface defines several system trace event types associated with control of and operation of the trace stream. This small set of system trace events includes the minimum required to interpret correctly the trace event information present in the stream. Other desirable system trace events for some particular application profile may be implemented and are encouraged; for example, process and thread scheduling, signal occurrence, and so on.

Each traced process shall have a mapping of the trace event names to trace event type identifiers that have been defined for that process. Each active trace stream shall have a mapping that incorporates all the trace event type identifiers predefined by the trace system plus all the mappings of trace event names to trace event type identifiers of the processes that are being traced into that trace stream. These mappings are defined from the instrumented application by calling the *posix_trace_eventid_open()* function and from the controller process by calling the *posix_trace_trid_eventid_open()* function. For a pre-recorded trace stream, the list of trace event types is obtained from the pre-recorded trace log.

The *st_ctime* and *st_mtime* fields of a file associated with an active trace stream shall be marked for update every time any of the tracing operations modifies that file.

The *st_atime* field of a file associated with a trace stream shall be marked for update every time any of the tracing operations causes data to be read from that file.

Results are undefined if the application performs any operation on a file descriptor associated with an active or pre-recorded trace stream until *posix_trace_shutdown()* or *posix_trace_close()* is called for that trace stream.

The main purpose of this option is to define a complete set of functions and concepts that allow a conforming application to be traced from creation to termination, whatever its realtime constraints and properties.

2.11.1 Tracing Data Definitions

2.11.1.1 Structures

The **<trace.h>** header shall define the *posix_trace_status_info* and *posix_trace_event_info* structures described below. Implementations may add extensions to these structures.

posix_trace_status_info Structure

To facilitate control of a trace stream, information about the current state of an active trace stream can be obtained dynamically. This structure is returned by a call to the *posix_trace_get_status()* function.

The **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following members:

Member Type	Member Name	Description
int	<i>posix_stream_status</i>	The operating mode of the trace stream.
int	<i>posix_stream_full_status</i>	The full status of the trace stream.
int	<i>posix_stream_overnrun_status</i>	Indicates whether trace events were lost in the trace stream.

If the Trace Log option is supported in addition to the Trace option, the **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following additional members:

Member Type	Member Name	Description
int	<i>posix_stream_flush_status</i>	Indicates whether a flush is in progress.
int	<i>posix_stream_flush_error</i>	Indicates whether any error occurred during the last flush operation.
int	<i>posix_log_overnrun_status</i>	Indicates whether trace events were lost in the trace log.
int	<i>posix_log_full_status</i>	The full status of the trace log.

The *posix_stream_status* member indicates the operating mode of the trace stream and shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_RUNNING

Tracing is in progress; that is, the trace stream is accepting trace events.

POSIX_TRACE_SUSPENDED

The trace stream is not accepting trace events. The tracing operation has not yet started or has stopped, either following a *posix_trace_stop()* function call or because the trace resources are exhausted.

The *posix_stream_full_status* member indicates the full status of the trace stream, and it shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_FULL

The space in the trace stream for trace events is exhausted.

POSIX_TRACE_NOT_FULL

There is still space available in the trace stream.

The combination of the *posix_stream_status* and *posix_stream_full_status* members also indicates the actual status of the stream. The status shall be interpreted as follows:

POSIX_TRACE_RUNNING and POSIX_TRACE_NOT_FULL

This status combination indicates that tracing is in progress, and there is space available for recording more trace events.

POSIX_TRACE_RUNNING and POSIX_TRACE_FULL

This status combination indicates that tracing is in progress and that the trace stream is full of trace events. This status combination cannot occur unless the *stream-full-policy* is set to

2980 POSIX_TRACE_LOOP. The trace stream contains trace events recorded during a moving
 2981 time window of prior trace events, and some older trace events may have been overwritten
 2982 and thus lost.

2983 POSIX_TRACE_SUSPENDED and POSIX_TRACE_NOT_FULL
 2984 This status combination indicates that tracing has not yet been started, has been stopped by
 2985 the *posix_trace_stop()* function, or has been cleared by the *posix_trace_clear()* function.

2986 POSIX_TRACE_SUSPENDED and POSIX_TRACE_FULL
 2987 This status combination indicates that tracing has been stopped by the implementation
 2988 because the *stream-full-policy* attribute was POSIX_TRACE_UNTIL_FULL and trace
 2989 resources were exhausted, or that the trace stream was stopped by the function
 2990 *posix_trace_stop()* at a time when trace resources were exhausted.

2991 The *posix_stream_outrun_status* member indicates whether trace events were lost in the trace
 2992 stream, and shall have one of the following values defined by manifest constants in the
 2993 **<trace.h>** header:

2994 POSIX_TRACE_OVERRUN
 2995 At least one trace event was lost and thus was not recorded in the trace stream.

2996 POSIX_TRACE_NO_OVERRUN
 2997 No trace events were lost.

2998 When the corresponding trace stream is created, the *posix_stream_outrun_status* member shall be
 2999 set to POSIX_TRACE_NO_OVERRUN.

3000 Whenever an overrun occurs, the *posix_stream_outrun_status* member shall be set to
 3001 POSIX_TRACE_OVERRUN.

3002 An overrun occurs when:

- 3003 • The policy is POSIX_TRACE_LOOP and a recorded trace event is overwritten.
- 3004 • The policy is POSIX_TRACE_UNTIL_FULL and the trace stream is full when a trace event is
 3005 generated.
- 3006 • If the Trace Log option is supported, the policy is POSIX_TRACE_FLUSH and at least one
 3007 trace event is lost while flushing the trace stream to the trace log.

3008 The *posix_stream_outrun_status* member is reset to zero after its value is read.

3009 If the Trace Log option is supported in addition to the Trace option, the *posix_stream_flush_status*,
 3010 *posix_stream_flush_error*, *posix_log_outrun_status*, and *posix_log_full_status* members are defined
 3011 as follows; otherwise, they are undefined.

3012 The *posix_stream_flush_status* member indicates whether a flush operation is being performed
 3013 and shall have one of the following values defined by manifest constants in the header
 3014 **<trace.h>**:

3015 POSIX_TRACE_FLUSHING
 3016 The trace stream is currently being flushed to the trace log.

3017 POSIX_TRACE_NOT_FLUSHING
 3018 No flush operation is in progress.

3019 The *posix_stream_flush_status* member shall be set to POSIX_TRACE_FLUSHING if a flush
 3020 operation is in progress either due to a call to the *posix_trace_flush()* function (explicit or caused
 3021 by a trace stream shutdown operation) or because the trace stream has become full with the
 3022 *stream-full-policy* attribute set to POSIX_TRACE_FLUSH. The *posix_stream_flush_status* member
 3023 shall be set to POSIX_TRACE_NOT_FLUSHING if no flush operation is in progress.

The *posix_stream_flush_error* member shall be set to zero if no error occurred during flushing. If an error occurred during a previous flushing operation, the *posix_stream_flush_error* member shall be set to the value of the first error that occurred. If more than one error occurs while flushing, error values after the first shall be discarded. The *posix_stream_flush_error* member is reset to zero after its value is read.

The *posix_log_outrun_status* member indicates whether trace events were lost in the trace log, and shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_OVERRUN

At least one trace event was lost.

POSIX_TRACE_NO_OVERRUN

No trace events were lost.

When the corresponding trace stream is created, the *posix_log_outrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN. Whenever an overrun occurs, this status shall be set to POSIX_TRACE_OVERRUN. The *posix_log_outrun_status* member is reset to zero after its value is read.

The *posix_log_full_status* member indicates the full status of the trace log, and it shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_FULL

The space in the trace log is exhausted.

POSIX_TRACE_NOT_FULL

There is still space available in the trace log.

The *posix_log_full_status* member is only meaningful if the *log-full-policy* attribute is either POSIX_TRACE_UNTIL_FULL or POSIX_TRACE_LOOP.

For an active trace stream without log, that is created by the *posix_trace_create()* function, the *posix_log_outrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN and the *posix_log_full_status* member shall be set to POSIX_TRACE_NOT_FULL.

posix_trace_event_info Structure

The trace event structure **posix_trace_event_info** contains the information for one recorded trace event. This structure is returned by the set of functions *posix_trace_getnext_event()*, *posix_trace_timedgetnext_event()*, and *posix_trace_trygetnext_event()*.

The **posix_trace_event_info** structure defined in **<trace.h>** shall contain at least the following members:

Member Type	Member Name	Description
trace_event_id_t	<i>posix_event_id</i>	Trace event type identification.
pid_t	<i>posix_pid</i>	Process ID of the process that generated the trace event.
void *	<i>posix_prog_address</i>	Address at which the trace point was invoked.
int	<i>posix_truncation_status</i>	Status about the truncation of the data associated with this trace event.
struct timespec	<i>posix_timestamp</i>	Time at which the trace event was generated.

In addition, if the Trace option and the Threads option are both supported, the **posix_trace_event_info** structure defined in **<trace.h>** shall contain the following additional member:

3068
3069
3070
3071
3072

Member Type	Member Name	Description
pthread_t	<i>posix_thread_id</i>	Thread ID of the thread that generated the trace event.

3073
3074
3075
3076
3077

The *posix_event_id* member represents the identification of the trace event type and its value is not directly defined by the user. This identification is returned by a call to one of the following functions: *posix_trace_trid_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*, or *posix_trace_eventid_open()*. The name of the trace event type can be obtained by calling *posix_trace_eventid_get_name()*.

3078
3079
3080

The *posix_pid* is the process identifier of the traced process which generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any process, the *posix_pid* member shall be set to zero.

3081
3082
3083
3084
3085

For a user trace event, the *posix_prog_address* member is the process mapped address of the point at which the associated call to the *posix_trace_event()* function was made. For a system trace event, if the trace event is caused by a system service explicitly called by the application, the *posix_prog_address* member shall be the address of the process at the point where the call to that system service was made.

3086
3087
3088
3089
3090

The *posix_truncation_status* member defines whether the data associated with a trace event has been truncated at the time the trace event was generated, or at the time the trace event was read from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event* argument from the *posix_trace_getnext_event()* function). The *posix_truncation_status* member shall have one of the following values defined by manifest constants in the **<trace.h>** header:

3091
3092

POSIX_TRACE_NOT_TRUNCATED

All the traced data is available.

3093
3094

POSIX_TRACE_TRUNCATED_RECORD

Data was truncated at the time the trace event was generated.

3095
3096
3097
3098

POSIX_TRACE_TRUNCATED_READ

Data was truncated at the time the trace event was read from a trace stream or a trace log because the reader's buffer was too small. This truncation status overrides the POSIX_TRACE_TRUNCATED_RECORD status.

3099
3100
3101

The *posix_timestamp* member shall be the time at which the trace event was generated. The clock used is implementation-defined, but the resolution of this clock can be retrieved by a call to the *posix_trace_attr_getclockres()* function.

3102

If the Threads option is supported in addition to the Trace option:

3103
3104
3105

- The *posix_thread_id* member is the identifier of the thread that generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any thread, the *posix_thread_id* member shall be set to zero.

3106

Otherwise, this member is undefined.

3107 2.11.1.2 Trace Stream Attributes

3108
3109

Trace streams have attributes that compose the **posix_trace_attr_t** trace stream attributes object. This object shall contain at least the following attributes:

3110

- The *generation-version* attribute identifies the origin and version of the trace system.

- 3111 • The *trace-name* attribute is a character string defined by the trace controller, and that
3112 identifies the trace stream.
- 3113 • The *creation-time* attribute represents the time of the creation of the trace stream.
- 3114 • The *clock-resolution* attribute defines the clock resolution of the clock used to generate
3115 timestamps.
- 3116 • The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly
3117 reserved for the trace events.
- 3118 • The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its
3119 value is `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or `POSIX_TRACE_FLUSH`.
- 3120 • The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

3121 In addition, if the Trace option and the Trace Inherit option are both supported, the
3122 **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following
3123 attributes:

- 3124 • The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing in
3125 its parent's process trace stream. It is either `POSIX_TRACE_INHERITED` or
3126 `POSIX_TRACE_CLOSE_FOR_CHILD`.

3127 In addition, if the Trace option and the Trace Log option are both supported, the
3128 **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following
3129 attribute:

- 3130 • If the file type corresponding to the trace log supports the `POSIX_TRACE_LOOP` or the
3131 `POSIX_TRACE_UNTIL_FULL` policies, the *log-max-size* attribute defines the maximum size
3132 in bytes of the trace log associated with an active trace stream. Other stream data—for
3133 example, trace attribute values—shall not be included in this size.
- 3134 • The *log-full-policy* attribute defines the policy of a trace log associated with an active trace
3135 stream to be `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or
3136 `POSIX_TRACE_APPEND`.

3137 2.11.2 Trace Event Type Definitions

3138 2.11.2.1 System Trace Event Type Definitions

3139 The following system trace event types, defined in the **<trace.h>** header, track the invocation of
3140 the trace operations:

- 3141 • `POSIX_TRACE_START` shall be associated with a trace start operation.
- 3142 • `POSIX_TRACE_STOP` shall be associated with a trace stop operation.
- 3143 • If the Trace Event Filter option is supported, `POSIX_TRACE_FILTER` shall be associated with
3144 a trace event type filter change operation.

3145 The following system trace event types, defined in the **<trace.h>** header, report operational trace
3146 events:

- 3147 • `POSIX_TRACE_OVERFLOW` shall mark the beginning of a trace overflow condition.
- 3148 • `POSIX_TRACE_RESUME` shall mark the end of a trace overflow condition.
- 3149 • If the Trace Log option is supported, `POSIX_TRACE_FLUSH_START` shall mark the
3150 beginning of a flush operation.

- If the Trace Log option is supported, POSIX_TRACE_FLUSH_STOP shall mark the end of a flush operation.

- If an implementation-defined trace error condition is reported, it shall be marked POSIX_TRACE_ERROR.

The interpretation of a trace stream or a trace log by a trace analyzer process relies on the information recorded for each trace event, and also on system trace events that indicate the invocation of trace control operations and trace system operational trace events.

The POSIX_TRACE_START and POSIX_TRACE_STOP trace events specify the time windows during which the trace stream is running.

- The POSIX_TRACE_STOP trace event with an associated data that is equal to zero indicates a call of the function *posix_trace_stop()*.
- The POSIX_TRACE_STOP trace event with an associated data that is different from zero indicates an automatic stop of the trace stream (see *posix_trace_attr_getstreamfullpolicy()*).

The POSIX_TRACE_FILTER trace event indicates that a trace event type filter value changed while the trace stream was running.

The POSIX_TRACE_ERROR serves to inform the analyzer process that an implementation-defined internal error of the trace system occurred.

The POSIX_TRACE_OVERFLOW trace event shall be reported with a timestamp equal to the timestamp of the first trace event overwritten. This is an indication that some generated trace events have been lost.

The POSIX_TRACE_RESUME trace event shall be reported with a timestamp equal to the timestamp of the first valid trace event reported after the overflow condition ends and shall be reported before this first valid trace event. This is an indication that the trace system is reliably recording trace events after an overflow condition.

Each of these trace event types shall be defined by a constant trace event name and a **trace_event_id_t** constant; trace event data is associated with some of these trace events.

If the Trace option is supported and the Trace Event Filter option and the Trace Log option are not supported, the following predefined system trace events in Table 2-3 shall be defined:

Table 2-3 Trace Option: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log option is not supported, the following predefined system trace events in Table 2-4 (on page 78) shall be defined:

Table 2-4 Trace and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	event_filter trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter new_event_filter trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter option is not supported, the following predefined system trace events in Table 2-5 shall be defined:

Table 2-5 Trace and Trace Log Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in Table 2-6 (on page 79) shall be defined:

Table 2-6 Trace, Trace Log, and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	event_filter trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter new_event_filter trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

2.11.2.2 User Trace Event Type Definitions

The user trace event `POSIX_TRACE_UNNAMED_USEREVENT` is defined in the `<trace.h>` header. If the limit of per-process user trace event names represented by `{TRACE_USER_EVENT_MAX}` has already been reached, this predefined user event shall be returned when the application tries to register more events than allowed. The data associated with this trace event is application-defined.

The following predefined user trace event in Table 2-7 shall be defined:

Table 2-7 Trace Option: User Trace Event

Event Name	Constant
posix_trace_unnamed_userevent	POSIX_TRACE_UNNAMED_USEREVENT

2.11.3 Trace Functions

The trace interface is built and structured to improve portability through use of trace data of opaque type. The object-oriented approach for the manipulation of trace attributes and trace event type identifiers requires definition of many constructor and selector functions which operate on these opaque types. Also, the trace interface must support several different tracing roles. To facilitate reading the trace interface, the trace functions are grouped into small functional sets supporting the three different roles:

- A trace controller process requires functions to set up and customize all the resources needed to run a trace stream, including:
 - Attribute initialization and destruction (`posix_trace_attr_init()`)
 - Identification information manipulation (`posix_trace_attr_getgenversion()`)
 - Trace system behavior modification (`posix_trace_attr_getinherited()`)
 - Trace stream and trace log size set (`posix_trace_attr_getmaxusereventsize()`)

- 3262 — Trace stream creation, flush, and shutdown (*posix_trace_create()*)
- 3263 — Trace stream and trace log clear (*posix_trace_clear()*)
- 3264 — Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)
- 3265 — Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)
- 3266 — Trace event type set manipulation (*posix_trace_eventset_empty()*)
- 3267 — Trace event type filter set (*posix_trace_set_filter()*)
- 3268 — Trace stream start and stop (*posix_trace_start()*)
- 3269 — Trace stream information and status read (*posix_trace_get_attr()*)
- 3270 • A traced process requires functions to instrument trace points:
- 3271 — Trace event type identifiers definition and trace points insertion (*posix_trace_event()*)
- 3272 • A trace analyzer process requires functions to retrieve information from a trace stream and
- 3273 trace log:
- 3274 — Identification information read (*posix_trace_attr_getgenversion()*)
- 3275 — Trace system behavior information read (*posix_trace_attr_getinherited()*)
- 3276 — Trace stream and trace log size get (*posix_trace_attr_getmaxusereventsized()*)
- 3277 — Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)
- 3278 — Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)
- 3279 — Trace log open, rewind, and close (*posix_trace_open()*)
- 3280 — Trace stream information and status read (*posix_trace_get_attr()*)
- 3281 — Trace event read (*posix_trace_getnext_event()*)

2.12 Data Types

All of the data types used by various functions are defined by the implementation. The following table describes some of these types. Other types referenced in the description of a function, not mentioned here, can be found in the appropriate header for that function.

Defined Type	Description
cc_t	Type used for terminal special characters.
clock_t	Integer or real-floating type used for processor times, as defined in the ISO C standard.
clockid_t	Used for clock ID type in some timer functions.
dev_t	Arithmetic type used for device numbers.
DIR	Type representing a directory stream.
div_t	Structure type returned by the <i>div()</i> function.
FILE	Structure containing information about a file.
glob_t	Structure type used in pathname pattern matching.
fpos_t	Type containing all information needed to specify uniquely every

3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346

Defined Type	Description
	position within a file.
gid_t	Integer type used for group IDs.
iconv_t	Type used for conversion descriptors.
id_t	Integer type used as a general identifier; can be used to contain at least the largest of a pid_t , uid_t , or gid_t .
ino_t	Unsigned integer type used for file serial numbers.
key_t	Arithmetic type used for XSI interprocess communication.
ldiv_t	Structure type returned by the <i>ldiv()</i> function.
mode_t	Integer type used for file attributes.
mqd_t	Used for message queue descriptors.
nfds_t	Integer type used for the number of file descriptors.
nlink_t	Integer type used for link counts.
off_t	Signed integer type used for file sizes.
pid_t	Signed integer type used for process and process group IDs.
pthread_attr_t	Used to identify a thread attribute object.
pthread_cond_t	Used for condition variables.
pthread_condattr_t	Used to identify a condition attribute object.
pthread_key_t	Used for thread-specific data keys.
pthread_mutex_t	Used for mutexes.
pthread_mutexattr_t	Used to identify a mutex attribute object.
pthread_once_t	Used for dynamic package initialization.
pthread_rwlock_t	Used for read-write locks.
pthread_rwlockattr_t	Used for read-write lock attributes.
pthread_t	Used to identify a thread.
ptrdiff_t	Signed integer type of the result of subtracting two pointers.
regex_t	Structure type used in regular expression matching.
regmatch_t	Structure type used in regular expression matching.
rlim_t	Unsigned integer type used for limit values, to which objects of type int and off_t can be cast without loss of value.
sem_t	Type used in performing semaphore operations.
sig_atomic_t	Integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
sigset_t	Integer or structure type of an object used to represent sets of signals.
size_t	Unsigned integer type used for size of objects.
speed_t	Type used for terminal baud rates.
ssize_t	Signed integer type used for a count of bytes or an error indication.
suseconds_t	Signed integer type used for time in microseconds.
tcflag_t	Type used for terminal modes.
time_t	Integer or real-floating type used for time in seconds, as defined in the ISO C standard.
timer_t	Used for timer ID returned by the <i>timer_create()</i> function.
uid_t	Integer type used for user IDs.
useconds_t	Unsigned integer type used for time in microseconds.
va_list	Type used for traversing variable argument lists.
wchar_t	Integer type whose range of values can represent distinct codes for

3347
3348
3349
3350
3351
3352
3353
3354
3355

Defined Type	Description
	all members of the largest extended character set specified by the supported locales.
wctype_t	Scalar type which represents a character class descriptor.
wint_t	Integer type capable of storing any valid value of wchar_t or WEOF.
wordexp_t	Structure type used in word expansion.

System Interfaces

3356

3357

3358

This chapter describes the functions, macros, and external variables to support applications portability at the C-language source level.

3359 **NAME**

3360 FD_CLR — macros for synchronous I/O multiplexing

3361 **SYNOPSIS**

3362 #include <sys/time.h>

3363 FD_CLR(int *fd*, fd_set **fdset*);3364 FD_ISSET(int *fd*, fd_set **fdset*);3365 FD_SET(int *fd*, fd_set **fdset*);3366 FD_ZERO(fd_set **fdset*);3367 **DESCRIPTION**3368 Refer to *pselect()*.

3369 **NAME**

3370 _Exit, _exit — terminate a process

3371 **SYNOPSIS**

3372 #include <stdlib.h> |

3373 void _Exit(int *status*); |

3374 #include <unistd.h>

3375 void _exit(int *status*);3376 **DESCRIPTION**3377 Refer to *exit()*.

3378 NAME

3379 _longjmp, _setjmp — non-local goto

3380 SYNOPSIS

```
3381 xSI      #include <setjmp.h>
3382
3382      void _longjmp(jmp_buf env, int val);
3383      int _setjmp(jmp_buf env);
3384
```

3385 DESCRIPTION

3386 The *_longjmp()* and *_setjmp()* functions shall be equivalent to *longjmp()* and *setjmp()*,
3387 respectively, with the additional restriction that *_longjmp()* and *_setjmp()* shall not manipulate
3388 the signal mask.

3389 If *_longjmp()* is called even though *env* was never initialized by a call to *_setjmp()*, or when the
3390 last such call was in a function that has since returned, the results are undefined.

3391 RETURN VALUE

3392 Refer to *longjmp()* and *setjmp()*.

3393 ERRORS

3394 No errors are defined.

3395 EXAMPLES

3396 None.

3397 APPLICATION USAGE

3398 If *_longjmp()* is executed and the environment in which *_setjmp()* was executed no longer exists,
3399 errors can occur. The conditions under which the environment of the *_setjmp()* no longer exists
3400 include exiting the function that contains the *_setjmp()* call, and exiting an inner block with
3401 temporary storage. This condition might not be detectable, in which case the *_longjmp()* occurs
3402 and, if the environment no longer exists, the contents of the temporary storage of an inner block
3403 are unpredictable. This condition might also cause unexpected process termination. If the
3404 function has returned, the results are undefined.

3405 Passing *longjmp()* a pointer to a buffer not created by *setjmp()*, passing *_longjmp()* a pointer to a
3406 buffer not created by *_setjmp()*, passing *siglongjmp()* a pointer to a buffer not created by
3407 *sigsetjmp()*, or passing any of these three functions a buffer that has been modified by the user
3408 can cause all the problems listed above, and more.

3409 The *_longjmp()* and *_setjmp()* functions are included to support programs written to historical
3410 system interfaces. New applications should use *siglongjmp()* and *sigsetjmp()* respectively.

3411 RATIONALE

3412 None.

3413 FUTURE DIRECTIONS

3414 The *_longjmp()* and *_setjmp()* functions may be marked LEGACY in a future version.

3415 SEE ALSO

3416 *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*, the Base Definitions volume of
3417 IEEE Std 1003.1-2001, <setjmp.h>

3418 CHANGE HISTORY

3419 First released in Issue 4, Version 2.

3420 **Issue 5**

3421 Moved from X/OPEN UNIX extension to BASE.

3422 NAME

3423 _toupper — transliterate uppercase characters to lowercase

3424 SYNOPSIS

3425 xSI #include <ctype.h>

3426 int _tolower(int c);

3427

3428 DESCRIPTION

3429 The *_tolower()* macro shall be equivalent to *tolower(c)* except that the application shall ensure
3430 that the argument *c* is an uppercase letter.

3431 RETURN VALUE

3432 Upon successful completion, *_tolower()* shall return the lowercase letter corresponding to the
3433 argument passed.

3434 ERRORS

3435 No errors are defined.

3436 EXAMPLES

3437 None.

3438 APPLICATION USAGE

3439 None.

3440 RATIONALE

3441 None.

3442 FUTURE DIRECTIONS

3443 None.

3444 SEE ALSO

3445 *tolower()*, *isupper()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
3446 <ctype.h>

3447 CHANGE HISTORY

3448 First released in Issue 1. Derived from Issue 1 of the SVID.

3449 Issue 6

3450 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

3451 NAME

3452 _toupper — transliterate lowercase characters to uppercase

3453 SYNOPSIS

3454 xSI #include <ctype.h>

3455 int _toupper(int c);

3456

3457 DESCRIPTION

3458 The *_toupper()* macro shall be equivalent to *toupper()* except that the application shall ensure
3459 that the argument *c* is a lowercase letter.

3460 RETURN VALUE

3461 Upon successful completion, *_toupper()* shall return the uppercase letter corresponding to the
3462 argument passed.

3463 ERRORS

3464 No errors are defined.

3465 EXAMPLES

3466 None.

3467 APPLICATION USAGE

3468 None.

3469 RATIONALE

3470 None.

3471 FUTURE DIRECTIONS

3472 None.

3473 SEE ALSO

3474 *islower()*, *toupper()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
3475 <ctype.h>

3476 CHANGE HISTORY

3477 First released in Issue 1. Derived from Issue 1 of the SVID.

3478 Issue 6

3479 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

3480 **NAME**

3481 a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

3482 **SYNOPSIS**

3483 xSI #include <stdlib.h>

3484 long a64l(const char *s);

3485 char *l64a(long value);

3486

3487 **DESCRIPTION**

3488 These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by
 3489 which 32-bit integers can be represented by up to six characters; each character represents a digit
 3490 in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall
 3491 be used for these operations.

3492 The characters used to represent digits are ' .' (dot) for 0, ' / ' for 1, ' 0 ' through ' 9 ' for [2,11],
 3493 ' A ' through ' Z ' for [12,37], and ' a ' through ' z ' for [38,63].

3494 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the
 3495 least significant, and return the corresponding **long** value. If the string pointed to by *s* contains
 3496 more than six characters, *a64l()* shall use the first six. If the first six characters of the string
 3497 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The
 3498 *a64l()* function shall scan the character string from left to right with the least significant digit on
 3499 the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than
 3500 32 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null
 3501 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

3502 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding radix-
 3503 64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

3504 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may
 3505 overwrite the buffer.

3506 The *l64a()* function need not be reentrant. A function that is not required to be reentrant is not
 3507 required to be thread-safe.

3508 **RETURN VALUE**

3509 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the
 3510 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

3511 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall
 3512 return a pointer to an empty string.

3513 **ERRORS**

3514 No errors are defined.

3515 **EXAMPLES**

3516 None.

3517 **APPLICATION USAGE**3518 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.3519 **RATIONALE**3520 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

3521 **FUTURE DIRECTIONS**

3522 None.

3523 **SEE ALSO**

3524 *strtoul()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**, the Shell and Utilities
3525 volume of IEEE Std 1003.1-2001, *uuencode*

3526 **CHANGE HISTORY**

3527 First released in Issue 4, Version 2.

3528 **Issue 5**

3529 Moved from X/OPEN UNIX extension to BASE.

3530 Normative text previously in the APPLICATION USAGE section is moved to the
3531 DESCRIPTION.

3532 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

3533 **NAME**

3534 abort — generate an abnormal process abort

3535 **SYNOPSIS**

3536 #include <stdlib.h>

3537 void abort(void);

3538 **DESCRIPTION**

3539 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 3540 conflict between the requirements described here and the ISO C standard is unintentional. This
 3541 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

3542 The *abort()* function shall cause abnormal process termination to occur, unless the signal
 3543 SIGABRT is being caught and the signal handler does not return.

3544 CX The abnormal termination processing shall include the default actions defined for SIGABRT and
 3545 may include an attempt to effect *fclose()* on all open streams.

3546 The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the
 3547 argument SIGABRT.

3548 CX The status made available to *wait()* or *waitpid()* by *abort()* shall be that of a process terminated
 3549 by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the SIGABRT
 3550 signal.

3551 **RETURN VALUE**3552 The *abort()* function shall not return.3553 **ERRORS**

3554 No errors are defined.

3555 **EXAMPLES**

3556 None.

3557 **APPLICATION USAGE**

3558 Catching the signal is intended to provide the application writer with a portable means to abort
 3559 processing, free from possible interference from any implementation-defined functions.

3560 **RATIONALE**

3561 The ISO/IEC 9899:1999 standard requires the *abort()* function to be async-signal-safe. Since
 3562 IEEE Std 1003.1-2001 defers to the ISO C standard, this required a change to the DESCRIPTION
 3563 from “shall include the effect of *fclose()*” to “may include an attempt to effect *fclose()*.”

3564 The revised wording permits some backwards-compatibility and avoids a potential deadlock
 3565 situation.

3566 The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded
 3567 paragraph from the DESCRIPTION:

3568 “On XSI-conformant systems, in addition the abnormal termination processing shall include the
 3569 effect of *fclose()* on message catalog descriptors.”

3570 There were several reasons to remove this paragraph:

- 3571 • No special processing of open message catalogs needs to be performed prior to abnormal
 3572 process termination.
- 3573 • The main reason to specifically mention that *abort()* includes the effect of *fclose()* on open
 3574 streams is to flush output queued on the stream. Message catalogs in this context are read-
 3575 only and, therefore, do not need to be flushed.

3576 • The effect of *fclose()* on a message catalog descriptor is unspecified. Message catalog
3577 descriptors are allowed, but not required to be implemented using a file descriptor, but there
3578 is no mention in IEEE Std 1003.1-2001 of a message catalog descriptor using a standard I/O
3579 stream FILE object as would be expected by *fclose()*.

3580 **FUTURE DIRECTIONS**

3581 None.

3582 **SEE ALSO**

3583 *exit()*, *kill()*, *raise()*, *signal()*, *wait()*, *waitpid()*, the Base Definitions volume of
3584 IEEE Std 1003.1-2001, **<stdlib.h>**

3585 **CHANGE HISTORY**

3586 First released in Issue 1. Derived from Issue 1 of the SVID.

3587 **Issue 6**

3588 Extensions beyond the ISO C standard are marked.

3589 Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

3590 The Open Group Base Resolution bwg2002-003 is applied.

3591 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the
3592 DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

3593 **NAME**

3594 abs — return an integer absolute value

3595 **SYNOPSIS**

3596 #include <stdlib.h>

3597 int abs(int i);

3598 **DESCRIPTION**

3599 cx The functionality described on this reference page is aligned with the ISO C standard. Any
3600 conflict between the requirements described here and the ISO C standard is unintentional. This
3601 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

3602 The *abs()* function shall compute the absolute value of its integer operand, *i*. If the result cannot
3603 be represented, the behavior is undefined.

3604 **RETURN VALUE**3605 The *abs()* function shall return the absolute value of its integer operand.3606 **ERRORS**

3607 No errors are defined.

3608 **EXAMPLES**

3609 None.

3610 **APPLICATION USAGE**

3611 In two's-complement representation, the absolute value of the negative integer with largest
3612 magnitude {INT_MIN} might not be representable.

3613 **RATIONALE**

3614 None.

3615 **FUTURE DIRECTIONS**

3616 None.

3617 **SEE ALSO**3618 *fabs()*, *labs()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>3619 **CHANGE HISTORY**

3620 First released in Issue 1. Derived from Issue 1 of the SVID.

3621 **Issue 6**

3622 Extensions beyond the ISO C standard are marked.

3623 **NAME**

3624 accept — accept a new connection on a socket

3625 **SYNOPSIS**

3626 #include <sys/socket.h>

```
3627           int accept(int socket, struct sockaddr *restrict address,
3628                      socklen_t *restrict address_len);
```

3629 **DESCRIPTION**

3630 The *accept()* function shall extract the first connection on the queue of pending connections,
 3631 create a new socket with the same socket type protocol and address family as the specified
 3632 socket, and allocate a new file descriptor for that socket.

3633 The *accept()* function takes the following arguments:

3634 *socket* Specifies a socket that was created with *socket()*, has been bound to an address
 3635 with *bind()*, and has issued a successful call to *listen()*.

3636 *address* Either a null pointer, or a pointer to a **sockaddr** structure where the address of
 3637 the connecting socket shall be returned.

3638 *address_len* Points to a **socklen_t** structure which on input specifies the length of the
 3639 supplied **sockaddr** structure, and on output specifies the length of the stored
 3640 address.

3641 If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored
 3642 in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in
 3643 the object pointed to by *address_len*.

3644 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
 3645 the stored address shall be truncated.

3646 If the protocol permits connections by unbound clients, and the peer is not bound, then the value
 3647 stored in the object pointed to by *address* is unspecified.

3648 If the listen queue is empty of connection requests and O_NONBLOCK is not set on the file
 3649 descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is
 3650 empty of connection requests and O_NONBLOCK is set on the file descriptor for the socket,
 3651 *accept()* shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].

3652 The accepted socket cannot itself accept more connections. The original socket remains open and
 3653 can accept more connections.

3654 **RETURN VALUE**

3655 Upon successful completion, *accept()* shall return the non-negative file descriptor of the accepted
 3656 socket. Otherwise, -1 shall be returned and *errno* set to indicate the error.

3657 **ERRORS**

3658 The *accept()* function shall fail if:

3659 [EAGAIN] or [EWOULDBLOCK]

3660 O_NONBLOCK is set for the socket file descriptor and no connections are
 3661 present to be accepted.

3662 [EBADF] The *socket* argument is not a valid file descriptor.

3663 [ECONNABORTED]

3664 A connection has been aborted.

3665	[EINTR]	The <i>accept()</i> function was interrupted by a signal that was caught before a valid connection arrived.
3666		
3667	[EINVAL]	The <i>socket</i> is not accepting connections.
3668	[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
3669	[ENFILE]	The maximum number of file descriptors in the system are already open.
3670	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
3671	[EOPNOTSUPP]	The <i>socket</i> type of the specified socket does not support accepting connections.
3672		
3673		The <i>accept()</i> function may fail if:
3674	[ENOBUFS]	No buffer space is available.
3675	[ENOMEM]	There was insufficient memory available to complete the operation.
3676	XSR [EPROTO]	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
3677		

3678 **EXAMPLES**

3679 None.

3680 **APPLICATION USAGE**

3681 When a connection is available, *select()* indicates that the file descriptor for the socket is ready
3682 for reading.

3683 **RATIONALE**

3684 None.

3685 **FUTURE DIRECTIONS**

3686 None.

3687 **SEE ALSO**

3688 *bind()*, *connect()*, *listen()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,
3689 <sys/socket.h>

3690 **CHANGE HISTORY**

3691 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

3692 The **restrict** keyword is added to the *accept()* prototype for alignment with the
3693 ISO/IEC 9899:1999 standard.

3694 **NAME**

3695 access — determine accessibility of a file

3696 **SYNOPSIS**

3697 #include <unistd.h>

3698 int access(const char *path, int amode);

3699 **DESCRIPTION**

3700 The *access()* function shall check the file named by the pathname pointed to by the *path*
 3701 argument for accessibility according to the bit pattern contained in *amode*, using the real user ID
 3702 in place of the effective user ID and the real group ID in place of the effective group ID.

3703 The value of *amode* is either the bitwise-inclusive OR of the access permissions to be checked
 3704 (R_OK, W_OK, X_OK) or the existence test (F_OK).

3705 If any access permissions are checked, each shall be checked individually, as described in the
 3706 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 3, Definitions. If the process has
 3707 appropriate privileges, an implementation may indicate success for X_OK even if none of the
 3708 execute file permission bits are set.

3709 **RETURN VALUE**

3710 If the requested access is permitted, *access()* succeeds and shall return 0; otherwise, -1 shall be
 3711 returned and *errno* shall be set to indicate the error.

3712 **ERRORS**3713 The *access()* function shall fail if:

3714 [EACCES] Permission bits of the file mode do not permit the requested access, or search
 3715 permission is denied on a component of the path prefix.

3716 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 3717 argument.

3718 [ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname
 3719 component is longer than {NAME_MAX}.
 3720

3721 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

3722 [ENOTDIR] A component of the path prefix is not a directory.

3723 [EROFS] Write access is requested for a file on a read-only file system.

3724 The *access()* function may fail if:

3725 [EINVAL] The value of the *amode* argument is invalid.

3726 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 3727 resolution of the *path* argument.

3728 [ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the *path* argument,
 3729 the length of the substituted pathname string exceeded {PATH_MAX}.
 3730

3731 [ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being
 3732 executed.

3733 **EXAMPLES**3734 **Testing for the Existence of a File**

3735 The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
3736 #include <unistd.h>
3737 ...
3738 int result;
3739 const char *filename = "/tmp/myfile";
3740 result = access (filename, F_OK);
```

3741 **APPLICATION USAGE**

3742 Additional values of *amode* other than the set defined in the description may be valid; for
 3743 example, if a system has extended access controls.

3744 **RATIONALE**

3745 In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()*
 3746 function because:

- 3747 1. Historical implementations of *access()* do not test file access correctly when the process'
 3748 real user ID is superuser. In particular, they always return zero when testing execute
 3749 permissions without regard to whether the file is executable.
- 3750 2. The superuser has complete access to all files on a system. As a consequence, programs
 3751 started by the superuser and switched to the effective user ID with lesser privileges cannot
 3752 use *access()* to test their file access permissions.

3753 However, the historical model of *eaccess()* does not resolve problem (1), so this volume of
 3754 IEEE Std 1003.1-2001 now allows *access()* to behave in the desired way because several
 3755 implementations have corrected the problem. It was also argued that problem (2) is more easily
 3756 solved by using *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the
 3757 error, rather than creating a new function that would not be as reliable. Therefore, *eaccess()* is not
 3758 included in this volume of IEEE Std 1003.1-2001.

3759 The sentence concerning appropriate privileges and execute permission bits reflects the two
 3760 possibilities implemented by historical implementations when checking superuser access for
 3761 X_OK.

3762 New implementations are discouraged from returning X_OK unless at least one execution
 3763 permission bit is set.

3764 **FUTURE DIRECTIONS**

3765 None.

3766 **SEE ALSO**

3767 *chmod()*, *stat()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

3768 **CHANGE HISTORY**

3769 First released in Issue 1. Derived from Issue 1 of the SVID.

3770 **Issue 6**

3771 The following new requirements on POSIX implementations derive from alignment with the
 3772 Single UNIX Specification:

- 3773 • The [ELOOP] mandatory error condition is added.
- 3774 • A second [ENAMETOOLONG] is added as an optional error condition.

- 3775 • The [ETXTBSY] optional error condition is added.

3776 The following changes were made to align with the IEEE P1003.1a draft standard:

- 3777 • The [ELOOP] optional error condition is added.

3778 **NAME**

3779 acos, acosf, acosl — arc cosine functions

3780 **SYNOPSIS**

3781 #include <math.h>

3782 double acos(double x);

3783 float acosf(float x);

3784 long double acosl(long double x);

3785 **DESCRIPTION**

3786 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 3787 conflict between the requirements described here and the ISO C standard is unintentional. This
 3788 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

3789 These functions shall compute the principal value of the arc cosine of their argument *x*. The
 3790 value of *x* should be in the range $[-1,1]$.

3791 An application wishing to check for error situations should set *errno* to zero and call
 3792 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 3793 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 3794 zero, an error has occurred.

3795 **RETURN VALUE**

3796 Upon successful completion, these functions shall return the arc cosine of *x*, in the range $[0,\pi]$
 3797 radians.

3798 **MX** For finite values of *x* not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 3799 supported), or an implementation-defined value shall be returned.

3800 **MX** If *x* is NaN, a NaN shall be returned.

3801 If *x* is +1, +0 shall be returned.

3802 If *x* is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 3803 defined value shall be returned.

3804 **ERRORS**

3805 These functions shall fail if:

3806 **MX** Domain Error The *x* argument is finite and is not in the range $[-1,1]$, or is $\pm\text{Inf}$.

3807 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 3808 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 3809 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 3810 shall be raised.

3811 **EXAMPLES**

3812 None.

3813 **APPLICATION USAGE**

3814 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 3815 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

3816 **RATIONALE**

3817 None.

3818 **FUTURE DIRECTIONS**

3819 None.

3820 **SEE ALSO**

3821 *cos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
3822 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

3823 **CHANGE HISTORY**

3824 First released in Issue 1. Derived from Issue 1 of the SVID.

3825 **Issue 5**

3826 The DESCRIPTION is updated to indicate how an application should check for an error. This
3827 text was previously published in the APPLICATION USAGE section.

3828 **Issue 6**3829 The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

3830 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
3831 revised to align with the ISO/IEC 9899:1999 standard.

3832 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
3833 marked.

3834 **NAME**

3835 acosh, acoshf, acoshl — inverse hyperbolic cosine functions

3836 **SYNOPSIS**

3837 #include <math.h>

3838 double acosh(double x);

3839 float acoshf(float x);

3840 long double acoshl(long double x);

3841 **DESCRIPTION**

3842 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 3843 conflict between the requirements described here and the ISO C standard is unintentional. This
 3844 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

3845 These functions shall compute the inverse hyperbolic cosine of their argument x .

3846 An application wishing to check for error situations should set *errno* to zero and call
 3847 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 3848 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 3849 zero, an error has occurred.

3850 **RETURN VALUE**

3851 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their
 3852 argument.

3853 MX For finite values of $x < 1$, a domain error shall occur, and either a NaN (if supported), or an
 3854 implementation-defined value shall be returned.

3855 MX If x is NaN, a NaN shall be returned.3856 If x is $+1$, $+0$ shall be returned.3857 If x is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

3858 If x is $-\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 3859 defined value shall be returned.

3860 **ERRORS**

3861 These functions shall fail if:

3862 MX Domain Error The x argument is finite and less than $+1.0$, or is $-\text{Inf}$.

3863 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 3864 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 3865 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 3866 shall be raised.

3867 **EXAMPLES**

3868 None.

3869 **APPLICATION USAGE**

3870 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 3871 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

3872 **RATIONALE**

3873 None.

3874 **FUTURE DIRECTIONS**

3875 None.

3876 **SEE ALSO**

3877 *cosh()*, *feclearexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section
3878 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

3879 **CHANGE HISTORY**

3880 First released in Issue 4, Version 2.

3881 **Issue 5**

3882 Moved from X/OPEN UNIX extension to BASE.

3883 **Issue 6**3884 The *acosh()* function is no longer marked as an extension.

3885 The *acoshf()* and *acoshl()* functions are added for alignment with the ISO/IEC 9899:1999
3886 standard.

3887 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
3888 revised to align with the ISO/IEC 9899:1999 standard.

3889 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
3890 marked.

3891 **NAME**

3892 acosl — arc cosine functions

3893 **SYNOPSIS**

3894 #include <math.h>

3895 long double acosl(long double x);

3896 **DESCRIPTION**3897 Refer to *acos()*.

3898 **NAME**3899 aio_cancel — cancel an asynchronous I/O request (**REALTIME**)3900 **SYNOPSIS**3901 AIO `#include <aio.h>`3902 `int aio_cancel(int fildes, struct aiocb *aiocbp);`

3903

3904 **DESCRIPTION**

3905 The *aio_cancel()* function shall attempt to cancel one or more asynchronous I/O requests
 3906 currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the
 3907 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then
 3908 all outstanding cancelable asynchronous I/O requests against *fildes* shall be canceled.

3909 Normal asynchronous notification shall occur for asynchronous I/O operations that are
 3910 successfully canceled. If there are requests that cannot be canceled, then the normal
 3911 asynchronous completion process shall take place for those requests when they are completed.

3912 For requested operations that are successfully canceled, the associated error status shall be set to
 3913 [ECANCELED] and the return status shall be -1. For requested operations that are not
 3914 successfully canceled, the *aiocbp* shall not be modified by *aio_cancel()*.

3915 If *aiocbp* is not NULL, then if *fildes* does not have the same value as the file descriptor with which
 3916 the asynchronous operation was initiated, unspecified results occur.

3917 Which operations are cancelable is implementation-defined.

3918 **RETURN VALUE**

3919 The *aio_cancel()* function shall return the value AIO_CANCELED to the calling process if the
 3920 requested operation(s) were canceled. The value AIO_NOTCANCELED shall be returned if at
 3921 least one of the requested operation(s) cannot be canceled because it is in progress. In this case,
 3922 the state of the other operations, if any, referenced in the call to *aio_cancel()* is not indicated by
 3923 the return value of *aio_cancel()*. The application may determine the state of affairs for these
 3924 operations by using *aio_error()*. The value AIO_ALLDONE is returned if all of the operations
 3925 have already completed. Otherwise, the function shall return -1 and set *errno* to indicate the
 3926 error.

3927 **ERRORS**

3928 The *aio_cancel()* function shall fail if:

3929 [EBADF] The *fildes* argument is not a valid file descriptor.

3930 **EXAMPLES**

3931 None.

3932 **APPLICATION USAGE**

3933 The *aio_cancel()* function is part of the Asynchronous Input and Output option and need not be
 3934 available on all implementations.

3935 **RATIONALE**

3936 None.

3937 **FUTURE DIRECTIONS**

3938 None.

3939 SEE ALSO

3940 *aio_read()*, *aio_write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

3941 CHANGE HISTORY

3942 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

3943 Issue 6

3944 The [ENOSYS] error condition has been removed as stubs need not be provided if an
3945 implementation does not support the Asynchronous Input and Output option.

3946 The APPLICATION USAGE section is added.

3947 NAME

3948 `aio_error` — retrieve errors status for an asynchronous I/O operation (**REALTIME**)

3949 SYNOPSIS

3950 AIO `#include <aio.h>`

3951 `int aio_error(const struct aiocb *aiocbp);`

3952

3953 DESCRIPTION

3954 The `aio_error()` function shall return the error status associated with the **aiocb** structure
 3955 referenced by the `aiocbp` argument. The error status for an asynchronous I/O operation is the
 3956 SIO `errno` value that would be set by the corresponding `read()`, `write()`, `fdatasync()`, or `fsync()`
 3957 operation. If the operation has not yet completed, then the error status shall be equal to
 3958 `[EINPROGRESS]`.

3959 RETURN VALUE

3960 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the
 3961 asynchronous operation has completed unsuccessfully, then the error status, as described for
 3962 SIO `read()`, `write()`, `fdatasync()`, and `fsync()`, shall be returned. If the asynchronous I/O operation has
 3963 not yet completed, then `[EINPROGRESS]` shall be returned.

3964 ERRORS

3965 The `aio_error()` function may fail if:

3966 `[EINVAL]` The `aiocbp` argument does not refer to an asynchronous operation whose
 3967 return status has not yet been retrieved.

3968 EXAMPLES

3969 None.

3970 APPLICATION USAGE

3971 The `aio_error()` function is part of the Asynchronous Input and Output option and need not be
 3972 available on all implementations.

3973 RATIONALE

3974 None.

3975 FUTURE DIRECTIONS

3976 None.

3977 SEE ALSO

3978 `aio_cancel()`, `aio_fsync()`, `aio_read()`, `aio_return()`, `aio_write()`, `close()`, `exec`, `exit()`, `fork()`, `lio_listio()`,
 3979 `lseek()`, `read()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

3980 CHANGE HISTORY

3981 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

3982 Issue 6

3983 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an
 3984 implementation does not support the Asynchronous Input and Output option.

3985 The APPLICATION USAGE section is added.

3986 **NAME**3987 aio_fsync — asynchronous file synchronization (**REALTIME**)3988 **SYNOPSIS**3989 AIO `#include <aio.h>`3990 `int aio_fsync(int op, struct aiocb *aiocbp);`

3991

3992 **DESCRIPTION**

3993 The *aio_fsync()* function shall asynchronously force all I/O operations associated with the file
 3994 indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp*
 3995 argument and queued at the time of the call to *aio_fsync()* to the synchronized I/O completion
 3996 state. The function call shall return when the synchronization request has been initiated or
 3997 queued to the file or device (even when the data cannot be synchronized immediately).

3998 If *op* is *O_DSYNC*, all currently queued I/O operations shall be completed as if by a call to
 3999 *fdatsync()*; that is, as defined for synchronized I/O data integrity completion. If *op* is *O_SYNC*,
 4000 all currently queued I/O operations shall be completed as if by a call to *fsync()*; that is, as
 4001 defined for synchronized I/O file integrity completion. If the *aio_fsync()* function fails, or if the
 4002 operation queued by *aio_fsync()* fails, then, as for *fsync()* and *fdatsync()*, outstanding I/O
 4003 operations are not guaranteed to have been completed.

4004 If *aio_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to
 4005 *aio_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of
 4006 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized
 4007 fashion.

4008 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used
 4009 as an argument to *aio_error()* and *aio_return()* in order to determine the error status and return
 4010 status, respectively, of the asynchronous operation while it is proceeding. When the request is
 4011 queued, the error status for the operation is *[EINPROGRESS]*. When all data has been
 4012 successfully transferred, the error status shall be reset to reflect the success or failure of the
 4013 operation. If the operation does not complete successfully, the error status for the operation shall
 4014 be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to
 4015 occur as specified in Section 2.4.1 (on page 28) when all operations have achieved synchronized
 4016 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the
 4017 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4018 completion, then the behavior is undefined.

4019 If the *aio_fsync()* function fails or *aiocbp* indicates an error condition, data is not guaranteed to
 4020 have been successfully transferred.

4021 **RETURN VALUE**

4022 The *aio_fsync()* function shall return the value 0 to the calling process if the I/O operation is
 4023 successfully queued; otherwise, the function shall return the value *-1* and set *errno* to indicate
 4024 the error.

4025 **ERRORS**4026 The *aio_fsync()* function shall fail if:

4027 *[EAGAIN]* The requested asynchronous operation was not queued due to temporary
 4028 resource limitations.

4029 *[EBADF]* The *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument
 4030 is not a valid file descriptor open for writing.

4031 [EINVAL] This implementation does not support synchronized I/O for this file.
4032 [EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified.
4033 In the event that any of the queued I/O operations fail, *aio_fsync()* shall return the error
4034 condition defined for *read()* and *write()*. The error is returned in the error status for the
4035 asynchronous *fsync()* operation, which can be retrieved using *aio_error()*.

4036 EXAMPLES

4037 None.

4038 APPLICATION USAGE

4039 The *aio_fsync()* function is part of the Asynchronous Input and Output option and need not be
4040 available on all implementations.

4041 RATIONALE

4042 None.

4043 FUTURE DIRECTIONS

4044 None.

4045 SEE ALSO

4046 *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*, the Base Definitions volume of
4047 IEEE Std 1003.1-2001, <**aio.h**>

4048 CHANGE HISTORY

4049 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4050 Issue 6

4051 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4052 implementation does not support the Asynchronous Input and Output option.

4053 The APPLICATION USAGE section is added.

4054 **NAME**4055 aio_read — asynchronous read from a file (**REALTIME**)4056 **SYNOPSIS**4057 AIO `#include <aio.h>`4058 `int aio_read(struct aiocb *aiocbp);`

4059

4060 **DESCRIPTION**

4061 The `aio_read()` function shall read `aiocbp->aio_nbytes` from the file associated with
 4062 `aiocbp->aio_fildes` into the buffer pointed to by `aiocbp->aio_buf`. The function call shall return when
 4063 the read request has been initiated or queued to the file or device (even when the data cannot be
 4064 delivered immediately).

4065 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 4066 at a priority equal to the scheduling priority of the process minus `aiocbp->aio_reqprio`.

4067 The `aiocbp` value may be used as an argument to `aio_error()` and `aio_return()` in order to
 4068 determine the error status and return status, respectively, of the asynchronous operation while it
 4069 is proceeding. If an error condition is encountered during queuing, the function call shall return
 4070 without having initiated or queued the request. The requested operation takes place at the
 4071 absolute position in the file as given by `aio_offset`, as if `lseek()` were called immediately prior to
 4072 the operation with an `offset` equal to `aio_offset` and a `whence` equal to `SEEK_SET`. After a
 4073 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file
 4074 is unspecified.

4075 The `aiocbp->aio_lio_opcode` field shall be ignored by `aio_read()`.

4076 The `aiocbp` argument points to an **aiocb** structure. If the buffer pointed to by `aiocbp->aio_buf` or
 4077 the control block pointed to by `aiocbp` becomes an illegal address prior to asynchronous I/O
 4078 completion, then the behavior is undefined.

4079 Simultaneous asynchronous operations using the same `aiocbp` produce undefined results.

4080 SIO If synchronized I/O is enabled on the file associated with `aiocbp->aio_fildes`, the behavior of this
 4081 function shall be according to the definitions of synchronized I/O data integrity completion and
 4082 synchronized I/O file integrity completion.

4083 For any system action that changes the process memory space while an asynchronous I/O is
 4084 outstanding to the address range being changed, the result of that action is undefined.

4085 For regular files, no data transfer shall occur past the offset maximum established in the open
 4086 file description associated with `aiocbp->aio_fildes`.

4087 **RETURN VALUE**

4088 The `aio_read()` function shall return the value zero to the calling process if the I/O operation is
 4089 successfully queued; otherwise, the function shall return the value `-1` and set `errno` to indicate
 4090 the error.

4091 **ERRORS**

4092 The `aio_read()` function shall fail if:

4093 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 4094 resource limitations.

4095 Each of the following conditions may be detected synchronously at the time of the call to
 4096 `aio_read()`, or asynchronously. If any of the conditions below are detected synchronously, the
 4097 `aio_read()` function shall return `-1` and set `errno` to the corresponding value. If any of the
 4098 conditions below are detected asynchronously, the return status of the asynchronous operation

4099 is set to -1, and the error status of the asynchronous operation is set to the corresponding value.

4100 [EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for reading.

4101 [EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid,
 4102 *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid
 4103 value.

4104 In the case that the *aio_read()* successfully queues the I/O operation but the operation is
 4105 subsequently canceled or encounters an error, the return status of the asynchronous operation is
 4106 one of the values normally returned by the *read()* function call. In addition, the error status of
 4107 the asynchronous operation is set to one of the error statuses normally set by the *read()* function
 4108 call, or one of the following values:

4109 [EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for reading.

4110 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 4111 *aio_cancel()* request.

4112 [EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid.

4113 The following condition may be detected synchronously or asynchronously:

4114 [EOVERFLOW] The file is a regular file, *aiocbp->aio_nbytes* is greater than 0, and the starting
 4115 offset in *aiocbp->aio_offset* is before the end-of-file and is at or beyond the offset
 4116 maximum in the open file description associated with *aiocbp->aio_fildes*.

4117 EXAMPLES

4118 None.

4119 APPLICATION USAGE

4120 The *aio_read()* function is part of the Asynchronous Input and Output option and need not be
 4121 available on all implementations.

4122 RATIONALE

4123 None.

4124 FUTURE DIRECTIONS

4125 None.

4126 SEE ALSO

4127 *aio_cancel()*, *aio_error()*, *lio_listio()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,
 4128 *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

4129 CHANGE HISTORY

4130 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4131 Large File Summit extensions are added.

4132 Issue 6

4133 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 4134 implementation does not support the Asynchronous Input and Output option.

4135 The APPLICATION USAGE section is added.

- 4136 The following new requirements on POSIX implementations derive from alignment with the
4137 Single UNIX Specification:
- 4138 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
4139 description. This change is to support large files.
 - 4140 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
4141 large files.

4142 **NAME**4143 aio_return — retrieve return status of an asynchronous I/O operation (**REALTIME**)4144 **SYNOPSIS**4145 AIO `#include <aio.h>`4146 `ssize_t aio_return(struct aiocb *aiocbp);`

4147

4148 **DESCRIPTION**

4149 The *aio_return()* function shall return the return status associated with the **aiocb** structure
 4150 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the
 4151 value that would be returned by the corresponding *read()*, *write()*, or *fsync()* function call. If the
 4152 error status for the operation is equal to [EINPROGRESS], then the return status for the
 4153 operation is undefined. The *aio_return()* function may be called exactly once to retrieve the
 4154 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in
 4155 a call to *aio_return()* or *aio_error()*, an error may be returned. When the **aiocb** structure referred
 4156 to by *aiocbp* is used to submit another asynchronous operation, then *aio_return()* may be
 4157 successfully used to retrieve the return status of that operation.

4158 **RETURN VALUE**

4159 If the asynchronous I/O operation has completed, then the return status, as described for *read()*,
 4160 *write()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed,
 4161 the results of *aio_return()* are undefined.

4162 **ERRORS**4163 The *aio_return()* function may fail if:

4164 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
 4165 return status has not yet been retrieved.

4166 **EXAMPLES**

4167 None.

4168 **APPLICATION USAGE**

4169 The *aio_return()* function is part of the Asynchronous Input and Output option and need not be
 4170 available on all implementations.

4171 **RATIONALE**

4172 None.

4173 **FUTURE DIRECTIONS**

4174 None.

4175 **SEE ALSO**

4176 *aio_cancel()*, *aio_error()*, *aio_fsync()*, *aio_read()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*,
 4177 *lseek()*, *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<aio.h>**

4178 **CHANGE HISTORY**

4179 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4180 **Issue 6**

4181 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 4182 implementation does not support the Asynchronous Input and Output option.

4183 The APPLICATION USAGE section is added.

4184 The [EINVAL] error condition is updated as a “may fail”. This is for consistency with the
 4185 DESCRIPTION.

NAME

aio_suspend — wait for an asynchronous I/O request (**REALTIME**)

SYNOPSIS

```
#include <aio.h>

int aio_suspend(const struct aiocb * const list[], int nent,
                const struct timespec *timeout);
```

DESCRIPTION

The *aio_suspend()* function shall suspend the calling thread until at least one of the asynchronous I/O operations referenced by the *list* argument has completed, until a signal interrupts the function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is, the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the function shall return without suspending the calling thread. The *list* argument is an array of pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of elements in the array. Each **aiocb** structure pointed to has been used in initiating an asynchronous I/O request via *aio_read()*, *aio_write()*, or *lio_listio()*. This array may contain NULL pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures that have not been used in submitting asynchronous I/O, the effect is undefined.

If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of the I/O operations referenced by *list* are completed, then *aio_suspend()* shall return with an error. If the Monotonic Clock option is supported, the clock that shall be used to measure this time interval shall be the CLOCK_MONOTONIC clock.

RETURN VALUE

If the *aio_suspend()* function returns after one or more asynchronous I/O operations have completed, the function shall return zero. Otherwise, the function shall return a value of -1 and set *errno* to indicate the error.

The application may determine which asynchronous I/O completed by scanning the associated error and return status using *aio_error()* and *aio_return()*, respectively.

ERRORS

The *aio_suspend()* function shall fail if:

[EAGAIN]	No asynchronous I/O indicated in the list referenced by <i>list</i> completed in the time interval indicated by <i>timeout</i> .
[EINTR]	A signal interrupted the <i>aio_suspend()</i> function. Note that, since each asynchronous I/O operation may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited.

EXAMPLES

None.

APPLICATION USAGE

The *aio_suspend()* function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

4230 FUTURE DIRECTIONS

4231 None.

4232 SEE ALSO

4233 *aio_read()*, *aio_write()*, *lio_listio()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

4234 CHANGE HISTORY

4235 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4236 Issue 6

4237 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4238 implementation does not support the Asynchronous Input and Output option.

4239 The APPLICATION USAGE section is added.

4240 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
4241 CLOCK_MONOTONIC clock, if supported, is used.

4242 **NAME**4243 aio_write — asynchronous write to a file (**REALTIME**)4244 **SYNOPSIS**4245 AIO `#include <aio.h>`4246 `int aio_write(struct aiocb *aiocbp);`

4247

4248 **DESCRIPTION**

4249 The *aio_write()* function shall write *aiocbp->aio_nbytes* to the file associated with *aiocbp->aio_fildes*
 4250 from the buffer pointed to by *aiocbp->aio_buf*. The function shall return when the write request
 4251 has been initiated or, at a minimum, queued to the file or device.

4252 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 4253 at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio*.

4254 The *aiocbp* argument may be used as an argument to *aio_error()* and *aio_return()* in order to
 4255 determine the error status and return status, respectively, of the asynchronous operation while it
 4256 is proceeding.

4257 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 4258 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4259 completion, then the behavior is undefined.

4260 If O_APPEND is not set for the file descriptor *aio_fildes*, then the requested operation shall take
 4261 place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called
 4262 immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to
 4263 SEEK_SET. If O_APPEND is set for the file descriptor, write operations append to the file in the
 4264 same order as the calls were made. After a successful call to enqueue an asynchronous I/O
 4265 operation, the value of the file offset for the file is unspecified.

4266 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_write()*.

4267 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4268 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this
 4269 function shall be according to the definitions of synchronized I/O data integrity completion, and
 4270 synchronized I/O file integrity completion.

4271 For any system action that changes the process memory space while an asynchronous I/O is
 4272 outstanding to the address range being changed, the result of that action is undefined.

4273 For regular files, no data transfer shall occur past the offset maximum established in the open
 4274 file description associated with *aiocbp->aio_fildes*.

4275 **RETURN VALUE**

4276 The *aio_write()* function shall return the value zero to the calling process if the I/O operation is
 4277 successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate
 4278 the error.

4279 **ERRORS**

4280 The *aio_write()* function shall fail if:

4281 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 4282 resource limitations.

4283 Each of the following conditions may be detected synchronously at the time of the call to
 4284 *aio_write()*, or asynchronously. If any of the conditions below are detected synchronously, the
 4285 *aio_write()* function shall return -1 and set *errno* to the corresponding value. If any of the

conditions below are detected asynchronously, the return status of the asynchronous operation shall be set to -1, and the error status of the asynchronous operation is set to the corresponding value.

[EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for writing.

[EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid, *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value.

In the case that the *aio_write()* successfully queues the I/O operation, the return status of the asynchronous operation shall be one of the values normally returned by the *write()* function call. If the operation is successfully queued but is subsequently canceled or encounters an error, the error status for the asynchronous operation contains one of the values normally set by the *write()* function call, or one of the following:

[EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for writing.

[EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid.

[ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel()* request.

The following condition may be detected synchronously or asynchronously:

[EFBIG] The file is a regular file, *aiocbp->aio_nbytes* is greater than 0, and the starting offset in *aiocbp->aio_offset* is at or beyond the offset maximum in the open file description associated with *aiocbp->aio_fildes*.

EXAMPLES

None.

APPLICATION USAGE

The *aio_write()* function is part of the Asynchronous Input and Output option and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel(), *aio_error()*, *aio_read()*, *aio_return()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*, *lseek()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Large File Summit extensions are added.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with

4329 *aiocbp->aio_fildes*.

- 4330 • The [EFBIG] error is added as part of the large file support extensions.

4331 **NAME**

4332 alarm — schedule an alarm signal

4333 **SYNOPSIS**

4334 #include <unistd.h>

4335 unsigned alarm(unsigned *seconds*);4336 **DESCRIPTION**

4337 The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after
 4338 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays
 4339 may prevent the process from handling the signal as soon as it is generated.

4340 If *seconds* is 0, a pending alarm request, if any, is canceled.

4341 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.
 4342 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time
 4343 at which the SIGALRM signal is generated.

4344 XSI Interactions between *alarm()* and any of *setitimer()*, *ualarm()*, or *usleep()* are unspecified.

4345 **RETURN VALUE**

4346 If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value
 4347 that is the number of seconds until the previous request would have generated a SIGALRM
 4348 signal. Otherwise, *alarm()* shall return 0.

4349 **ERRORS**

4350 The *alarm()* function is always successful, and no return value is reserved to indicate an error.

4351 **EXAMPLES**

4352 None.

4353 **APPLICATION USAGE**

4354 The *fork()* function clears pending alarms in the child process. A new process image created by
 4355 one of the *exec* functions inherits the time left to an alarm signal in the old process' image.

4356 Application writers should note that the type of the argument *seconds* and the return value of
 4357 *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces
 4358 Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX},
 4359 which the ISO C standard sets as 65535, and any application passing a larger value is restricting
 4360 its portability. A different type was considered, but historical implementations, including those
 4361 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

4362 Application writers should be aware of possible interactions when the same process uses both
 4363 the *alarm()* and *sleep()* functions.

4364 **RATIONALE**

4365 Many historical implementations (including Version 7 and System V) allow an alarm to occur up
 4366 to a second early. Other implementations allow alarms up to half a second or one clock tick
 4367 early or do not allow them to occur early at all. The latter is considered most appropriate, since it
 4368 gives the most predictable behavior, especially since the signal can always be delayed for an
 4369 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument
 4370 as the minimum amount of time they wish to have elapse before the signal.

4371 The term “realtime” here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time
 4372 as common English usage, and has nothing to do with “realtime operating systems”. It is in
 4373 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

4374 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are
 4375 silently rounded down to an implementation-defined maximum value. This maximum is large

4376 enough (to the order of several months) that the effect is not noticeable.

4377 There were two possible choices for alarm generation in multi-threaded applications: generation
4378 for the calling thread or generation for the process. The first option would not have been
4379 particularly useful since the alarm state is maintained on a per-process basis and the alarm that
4380 is established by the last invocation of *alarm()* is the only one that would be active.

4381 Furthermore, allowing generation of an asynchronous signal for a thread would have introduced
4382 an exception to the overall signal model. This requires a compelling reason in order to be
4383 justified.

4384 **FUTURE DIRECTIONS**

4385 None.

4386 **SEE ALSO**

4387 *alarm()*, *exec*, *fork()*, *getitimer()*, *pause()*, *sigaction()*, *sleep()*, *ualarm()*, *usleep()*, the Base
4388 Definitions volume of IEEE Std 1003.1-2001, <**signal.h**>, <**unistd.h**>

4389 **CHANGE HISTORY**

4390 First released in Issue 1. Derived from Issue 1 of the SVID.

4391 **Issue 6**

4392 The following new requirements on POSIX implementations derive from alignment with the
4393 Single UNIX Specification:

- 4394 • The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*, and
4395 *usleep()* functions are unspecified.

4396 **NAME**

4397 asctime, asctime_r — convert date and time to a string

4398 **SYNOPSIS**

4399 #include <time.h>

4400 char *asctime(const struct tm *timeptr);

4401 TSF char *asctime_r(const struct tm *restrict tm, char *restrict buf);

4402

4403 **DESCRIPTION**

4404 CX For *asctime()*: The functionality described on this reference page is aligned with the ISO C
 4405 standard. Any conflict between the requirements described here and the ISO C standard is
 4406 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4407 The *asctime()* function shall convert the broken-down time in the structure pointed to by *timeptr*
 4408 into a string in the form:

4409 Sun Sep 16 01:03:52 1973\n\0

4410 using the equivalent of the following algorithm:

```

4411 char *asctime(const struct tm *timeptr)
4412 {
4413     static char wday_name[7][3] = {
4414         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
4415     };
4416     static char mon_name[12][3] = {
4417         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
4418         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
4419     };
4420     static char result[26];
4421     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
4422         wday_name[timeptr->tm_wday],
4423         mon_name[timeptr->tm_mon],
4424         timeptr->tm_mday, timeptr->tm_hour,
4425         timeptr->tm_min, timeptr->tm_sec,
4426         1900 + timeptr->tm_year);
4427     return result;
4428 }
```

4429 The **tm** structure is defined in the <time.h> header.

4430 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 4431 objects: a broken-down time structure and an array of type **char**. Execution of any of the
 4432 functions may overwrite the information returned in either of these objects by any of the other
 4433 functions.

4434 The *asctime()* function need not be reentrant. A function that is not required to be reentrant is not
 4435 required to be thread-safe.

4436 TSF The *asctime_r()* function shall convert the broken-down time in the structure pointed to by *tm*
 4437 into a string (of the same form as that returned by *asctime()*) that is placed in the user-supplied
 4438 buffer pointed to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

4439 **RETURN VALUE**

4440 Upon successful completion, *asctime()* shall return a pointer to the string.

4441 TSF Upon successful completion, *asctime_r()* shall return a pointer to a character string containing
4442 the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful,
4443 it shall return NULL.

4444 **ERRORS**

4445 No errors are defined.

4446 **EXAMPLES**

4447 None.

4448 **APPLICATION USAGE**

4449 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.
4450 This function is included for compatibility with older implementations, and does not support
4451 localized date and time formats. Applications should use *strftime()* to achieve maximum
4452 portability.

4453 The *asctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead
4454 of possibly using a static data area that may be overwritten by each call.

4455 **RATIONALE**

4456 None.

4457 **FUTURE DIRECTIONS**

4458 None.

4459 **SEE ALSO**

4460 *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
4461 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

4462 **CHANGE HISTORY**

4463 First released in Issue 1. Derived from Issue 1 of the SVID.

4464 **Issue 5**

4465 Normative text previously in the APPLICATION USAGE section is moved to the
4466 DESCRIPTION.

4467 The *asctime_r()* function is included for alignment with the POSIX Threads Extension.

4468 A note indicating that the *asctime()* function need not be reentrant is added to the
4469 DESCRIPTION.

4470 **Issue 6**

4471 The *asctime_r()* function is marked as part of the Thread-Safe Functions option.

4472 Extensions beyond the ISO C standard are marked.

4473 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
4474 its avoidance of possibly using a static data area.

4475 The DESCRIPTION of *asctime_r()* is updated to describe the format of the string returned.

4476 The **restrict** keyword is added to the *asctime_r()* prototype for alignment with the
4477 ISO/IEC 9899:1999 standard.

4478 **NAME**

4479 asin, asinf, asinl — arc sine function

4480 **SYNOPSIS**

4481 #include <math.h>

4482 double asin(double x);

4483 float asinf(float x);

4484 long double asinl(long double x);

4485 **DESCRIPTION**

4486 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 4487 conflict between the requirements described here and the ISO C standard is unintentional. This
 4488 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4489 These functions shall compute the principal value of the arc sine of their argument *x*. The value
 4490 of *x* should be in the range $[-1,1]$.

4491 An application wishing to check for error situations should set *errno* to zero and call
 4492 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 4493 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 4494 zero, an error has occurred.

4495 **RETURN VALUE**

4496 Upon successful completion, these functions shall return the arc sine of *x*, in the range
 4497 $[-\pi/2, \pi/2]$ radians.

4498 MX For finite values of *x* not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 4499 supported), or an implementation-defined value shall be returned.

4500 MX If *x* is NaN, a NaN shall be returned.

4501 If *x* is ± 0 , *x* shall be returned.

4502 If *x* is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 4503 defined value shall be returned.

4504 If *x* is subnormal, a range error may occur and *x* should be returned.

4505 **ERRORS**

4506 These functions shall fail if:

4507 MX Domain Error The *x* argument is finite and is not in the range $[-1,1]$, or is $\pm \text{Inf}$.

4508 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4509 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 4510 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 4511 shall be raised.

4512 These functions may fail if:

4513 MX Range Error The value of *x* is subnormal.

4514 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4515 then *errno* shall be set to [ERANGE]. If the integer expression
 4516 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 4517 floating-point exception shall be raised.

4518 **EXAMPLES**

4519 None.

4520 **APPLICATION USAGE**

4521 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
4522 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

4523 **RATIONALE**

4524 None.

4525 **FUTURE DIRECTIONS**

4526 None.

4527 **SEE ALSO**

4528 *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, the Base Definitions volume of IEEE Std 1003.1-2001,
4529 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

4530 **CHANGE HISTORY**

4531 First released in Issue 1. Derived from Issue 1 of the SVID.

4532 **Issue 5**

4533 The DESCRIPTION is updated to indicate how an application should check for an error. This
4534 text was previously published in the APPLICATION USAGE section.

4535 **Issue 6**4536 The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

4537 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
4538 revised to align with the ISO/IEC 9899:1999 standard.

4539 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
4540 marked.

4541 **NAME**

4542 asinh, asinhf, asinhl — inverse hyperbolic sine functions

4543 **SYNOPSIS**

4544 #include <math.h>

4545 double asinh(double x);

4546 float asinhf(float x);

4547 long double asinhl(long double x);

4548 **DESCRIPTION**

4549 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 4550 conflict between the requirements described here and the ISO C standard is unintentional. This
 4551 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4552 These functions shall compute the inverse hyperbolic sine of their argument *x*.

4553 An application wishing to check for error situations should set *errno* to zero and call
 4554 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 4555 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 4556 zero, an error has occurred.

4557 **RETURN VALUE**

4558 Upon successful completion, these functions shall return the inverse hyperbolic sine of their
 4559 argument.

4560 MX If *x* is NaN, a NaN shall be returned.4561 If *x* is ± 0 , or $\pm \text{Inf}$, *x* shall be returned.4562 If *x* is subnormal, a range error may occur and *x* should be returned.4563 **ERRORS**

4564 These functions may fail if:

4565 MX Range Error The value of *x* is subnormal.

4566 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4567 then *errno* shall be set to [ERANGE]. If the integer expression
 4568 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 4569 floating-point exception shall be raised.

4570 **EXAMPLES**

4571 None.

4572 **APPLICATION USAGE**

4573 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 4574 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

4575 **RATIONALE**

4576 None.

4577 **FUTURE DIRECTIONS**

4578 None.

4579 **SEE ALSO**

4580 *feclearexcept*(), *fetestexcept*(), *sinh*(), the Base Definitions volume of IEEE Std 1003.1-2001, Section
 4581 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

4582 **CHANGE HISTORY**

4583 First released in Issue 4, Version 2.

4584 **Issue 5**

4585 Moved from X/OPEN UNIX extension to BASE.

4586 **Issue 6**4587 The *asinh()* function is no longer marked as an extension.4588 The *asinhf()* and *asinhll()* functions are added for alignment with the ISO/IEC 9899:1999
4589 standard.4590 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
4591 revised to align with the ISO/IEC 9899:1999 standard.4592 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
4593 marked.

4594 **NAME**4595 *asinl* — arc sine function4596 **SYNOPSIS**4597 `#include <math.h>`4598 `long double asinl(long double x);`4599 **DESCRIPTION**4600 Refer to *asin()*.

4601 **NAME**

4602 assert — insert program diagnostics

4603 **SYNOPSIS**

4604 #include <assert.h>

4605 void assert(*scalar expression*);4606 **DESCRIPTION**

4607 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 4608 conflict between the requirements described here and the ISO C standard is unintentional. This
 4609 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4610 The *assert()* macro shall insert diagnostics into programs; it shall expand to a **void** expression.
 4611 When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal
 4612 to 0), *assert()* shall write information about the particular call that failed on *stderr* and shall call
 4613 *abort()*.

4614 The information written about the call that failed shall include the text of the argument, the
 4615 name of the source file, the source file line number, and the name of the enclosing function; the
 4616 latter are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__` and of
 4617 the identifier `__func__`.

4618 Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the
 4619 preprocessor control statement **#define** `NDEBUG` ahead of the **#include** `<assert.h>` statement,
 4620 shall stop assertions from being compiled into the program.

4621 **RETURN VALUE**4622 The *assert()* macro shall not return a value.4623 **ERRORS**

4624 No errors are defined.

4625 **EXAMPLES**

4626 None.

4627 **APPLICATION USAGE**

4628 None.

4629 **RATIONALE**

4630 None.

4631 **FUTURE DIRECTIONS**

4632 None.

4633 **SEE ALSO**4634 *abort()*, *stderr*, the Base Definitions volume of IEEE Std 1003.1-2001, `<assert.h>`4635 **CHANGE HISTORY**

4636 First released in Issue 1. Derived from Issue 1 of the SVID.

4637 **Issue 6**

4638 The prototype for the *expression* argument to *assert()* is changed from **int** to **scalar** for alignment
 4639 with the ISO / IEC 9899:1999 standard.

4640 The DESCRIPTION of *assert()* is updated for alignment with the ISO / IEC 9899:1999 standard.

4641 **NAME**

4642 atan, atanf, atanl — arc tangent function

4643 **SYNOPSIS**

4644 #include <math.h>

4645 double atan(double x);

4646 float atanf(float x);

4647 long double atanl(long double x);

4648 **DESCRIPTION**

4649 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 4650 conflict between the requirements described here and the ISO C standard is unintentional. This
 4651 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4652 These functions shall compute the principal value of the arc tangent of their argument *x*.

4653 An application wishing to check for error situations should set *errno* to zero and call
 4654 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 4655 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 4656 zero, an error has occurred.

4657 **RETURN VALUE**

4658 Upon successful completion, these functions shall return the arc tangent of *x* in the range
 4659 $[-\pi/2, \pi/2]$ radians.

4660 MX If *x* is NaN, a NaN shall be returned.4661 If *x* is ± 0 , *x* shall be returned.4662 If *x* is $\pm \text{Inf}$, $\pm \pi/2$ shall be returned.4663 If *x* is subnormal, a range error may occur and *x* should be returned.4664 **ERRORS**

4665 These functions may fail if:

4666 MX Range Error The value of *x* is subnormal.

4667 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4668 then *errno* shall be set to [ERANGE]. If the integer expression
 4669 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 4670 floating-point exception shall be raised.

4671 **EXAMPLES**

4672 None.

4673 **APPLICATION USAGE**

4674 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 4675 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

4676 **RATIONALE**

4677 None.

4678 **FUTURE DIRECTIONS**

4679 None.

4680 **SEE ALSO**

4681 *atan2()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*, the Base Definitions volume of
 4682 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
 4683 <math.h>

4684 **CHANGE HISTORY**

4685 First released in Issue 1. Derived from Issue 1 of the SVID.

4686 **Issue 5**4687 The DESCRIPTION is updated to indicate how an application should check for an error. This
4688 text was previously published in the APPLICATION USAGE section.4689 **Issue 6**4690 The *atanf()* and *atanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.4691 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
4692 revised to align with the ISO/IEC 9899:1999 standard.4693 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
4694 marked.

4695 **NAME**

4696 atan2, atan2f, atan2l — arc tangent functions

4697 **SYNOPSIS**

4698 #include <math.h>

4699 double atan2(double y, double x);

4700 float atan2f(float y, float x);

4701 long double atan2l(long double y, long double x);

4702 **DESCRIPTION**

4703 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 4704 conflict between the requirements described here and the ISO C standard is unintentional. This
 4705 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4706 These functions shall compute the principal value of the arc tangent of y/x , using the signs of
 4707 both arguments to determine the quadrant of the return value.

4708 An application wishing to check for error situations should set *errno* to zero and call
 4709 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 4710 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 4711 zero, an error has occurred.

4712 **RETURN VALUE**

4713 Upon successful completion, these functions shall return the arc tangent of y/x in the range
 4714 $[-\pi, \pi]$ radians.

4715 If y is ± 0 and x is < 0 , $\pm\pi$ shall be returned.4716 If y is ± 0 and x is > 0 , ± 0 shall be returned.4717 If y is < 0 and x is ± 0 , $-\pi/2$ shall be returned.4718 If y is > 0 and x is ± 0 , $\pi/2$ shall be returned.4719 If x is 0, a pole error shall not occur.4720 MX If either x or y is NaN, a NaN shall be returned.4721 If the result underflows, a range error may occur and y/x should be returned.4722 If y is ± 0 and x is -0 , $\pm\pi$ shall be returned.4723 If y is ± 0 and x is $+0$, ± 0 shall be returned.4724 For finite values of $\pm y > 0$, if x is $-\text{Inf}$, $\pm\pi$ shall be returned.4725 For finite values of $\pm y > 0$, if x is $+\text{Inf}$, ± 0 shall be returned.4726 For finite values of x , if y is $\pm\text{Inf}$, $\pm\pi/2$ shall be returned.4727 If y is $\pm\text{Inf}$ and x is $-\text{Inf}$, $\pm 3\pi/4$ shall be returned.4728 If y is $\pm\text{Inf}$ and x is $+\text{Inf}$, $\pm\pi/4$ shall be returned.

4729 If both arguments are 0, a domain error shall not occur.

4730 **ERRORS**

4731 These functions may fail if:

4732 MX Range Error The result underflows.

4733 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4734 then *errno* shall be set to [ERANGE]. If the integer expression

4735 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
4736 floating-point exception shall be raised.

4737 EXAMPLES

4738 None.

4739 APPLICATION USAGE

4740 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
4741 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

4742 RATIONALE

4743 None.

4744 FUTURE DIRECTIONS

4745 None.

4746 SEE ALSO

4747 *atan()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*, the Base Definitions volume of
4748 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
4749 **<math.h>**

4750 CHANGE HISTORY

4751 First released in Issue 1. Derived from Issue 1 of the SVID.

4752 Issue 5

4753 The DESCRIPTION is updated to indicate how an application should check for an error. This
4754 text was previously published in the APPLICATION USAGE section.

4755 Issue 6

4756 The *atan2f()* and *atan2l()* functions are added for alignment with the ISO/IEC 9899:1999
4757 standard.

4758 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
4759 revised to align with the ISO/IEC 9899:1999 standard.

4760 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
4761 marked.

4762 **NAME**

4763 atanf — arc tangent function

4764 **SYNOPSIS**

4765 #include <math.h>

4766 float atanf(float x);

4767 **DESCRIPTION**4768 Refer to *atan()*.

4769 **NAME**

4770 atanh, atanhf, atanhf — inverse hyperbolic tangent functions

4771 **SYNOPSIS**

4772 #include <math.h>

4773 double atanh(double x);

4774 float atanhf(float x);

4775 long double atanhf(long double x);

4776 **DESCRIPTION**

4777 CX The functionality described on this reference page is aligned with the ISO C standard. Any
4778 conflict between the requirements described here and the ISO C standard is unintentional. This
4779 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4780 These functions shall compute the inverse hyperbolic tangent of their argument x .

4781 An application wishing to check for error situations should set *errno* to zero and call
4782 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
4783 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
4784 zero, an error has occurred.

4785 **RETURN VALUE**

4786 Upon successful completion, these functions shall return the inverse hyperbolic tangent of their
4787 argument.

4788 If x is ± 1 , a pole error shall occur, and *atanh()*, *atanhf()*, and *atanf()* shall return the value of the
4789 macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the
4790 correct value of the function.

4791 MX For finite $|x| > 1$, a domain error shall occur, and either a NaN (if supported), or an
4792 implementation-defined value shall be returned.

4793 MX If x is NaN, a NaN shall be returned.

4794 If x is ± 0 , x shall be returned.

4795 If x is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
4796 defined value shall be returned.

4797 If x is subnormal, a range error may occur and x should be returned.

4798 **ERRORS**

4799 These functions shall fail if:

4800 MX Domain Error The x argument is finite and not in the range $[-1, 1]$, or is $\pm \text{Inf}$.

4801 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
4802 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
4803 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
4804 shall be raised.

4805 Pole Error The x argument is ± 1 .

4806 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
4807 then *errno* shall be set to [ERANGE]. If the integer expression
4808 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
4809 zero floating-point exception shall be raised.

4810 These functions may fail if:

4811 MX **Range Error** The value of x is subnormal.

4812 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 4813 then *errno* shall be set to [ERANGE]. If the integer expression
 4814 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow
 4815 floating-point exception shall be raised.

4816 **EXAMPLES**

4817 None.

4818 **APPLICATION USAGE**

4819 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`
 4820 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

4821 **RATIONALE**

4822 None.

4823 **FUTURE DIRECTIONS**

4824 None.

4825 **SEE ALSO**

4826 *feclearexcept()*, *fetestexcept()*, *tanh()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section
 4827 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

4828 **CHANGE HISTORY**

4829 First released in Issue 4, Version 2.

4830 **Issue 5**

4831 Moved from X/OPEN UNIX extension to BASE.

4832 **Issue 6**

4833 The *atanh()* function is no longer marked as an extension.

4834 The *atanhf()* and *atanhl()* functions are added for alignment with the ISO/IEC 9899:1999
 4835 standard.

4836 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 4837 revised to align with the ISO/IEC 9899:1999 standard.

4838 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 4839 marked.

4840 **NAME**

4841 atanl — arc tangent function

4842 **SYNOPSIS**

4843 #include <math.h>

4844 long double atanl(long double x);

4845 **DESCRIPTION**4846 Refer to *atan()*.

4847 **NAME**

4848 atexit — register a function to run at process termination

4849 **SYNOPSIS**

4850 #include <stdlib.h>

4851 int atexit(void (*func)(void));

4852 **DESCRIPTION**4853 CX The functionality described on this reference page is aligned with the ISO C standard. Any
4854 conflict between the requirements described here and the ISO C standard is unintentional. This
4855 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.4856 The *atexit()* function shall register the function pointed to by *func*, to be called without
4857 arguments at normal program termination. At normal program termination, all functions
4858 registered by the *atexit()* function shall be called, in the reverse order of their registration, except
4859 that a function is called after any previously registered functions that had already been called at
4860 the time it was registered. Normal termination occurs either by a call to *exit()* or a return from
4861 *main()*.4862 At least 32 functions can be registered with *atexit()*.4863 CX After a successful call to any of the *exec* functions, any functions previously registered by *atexit()*
4864 shall no longer be registered.4865 **RETURN VALUE**4866 Upon successful completion, *atexit()* shall return 0; otherwise, it shall return a non-zero value.4867 **ERRORS**

4868 No errors are defined.

4869 **EXAMPLES**

4870 None.

4871 **APPLICATION USAGE**4872 The functions registered by a call to *atexit()* must return to ensure that all registered functions
4873 are called.4874 The application should call *sysconf()* to obtain the value of {ATEXIT_MAX}, the number of
4875 functions that can be registered. There is no way for an application to tell how many functions
4876 have already been registered with *atexit()*.4877 **RATIONALE**

4878 None.

4879 **FUTURE DIRECTIONS**

4880 None.

4881 **SEE ALSO**4882 *exit()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>4883 **CHANGE HISTORY**

4884 First released in Issue 4. Derived from the ANSI C standard.

4885 **Issue 6**

4886 Extensions beyond the ISO C standard are marked.

4887 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

4888 **NAME**4889 `atof` — convert a string to a double-precision number4890 **SYNOPSIS**4891 `#include <stdlib.h>`4892 `double atof(const char *str);`4893 **DESCRIPTION**4894 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any
4895 conflict between the requirements described here and the ISO C standard is unintentional. This
4896 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.4897 The call `atof(str)` shall be equivalent to:4898 `strtod(str, (char **)NULL),`4899 except that the handling of errors may differ. If the value cannot be represented, the behavior is
4900 undefined.4901 **RETURN VALUE**4902 The `atof()` function shall return the converted value if the value can be represented.4903 **ERRORS**

4904 No errors are defined.

4905 **EXAMPLES**

4906 None.

4907 **APPLICATION USAGE**4908 The `atof()` function is subsumed by `strtod()` but is retained because it is used extensively in
4909 existing code. If the number is not known to be in range, `strtod()` should be used because `atof()` is
4910 not required to perform any error checking.4911 **RATIONALE**

4912 None.

4913 **FUTURE DIRECTIONS**

4914 None.

4915 **SEE ALSO**4916 `strtod()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`4917 **CHANGE HISTORY**

4918 First released in Issue 1. Derived from Issue 1 of the SVID.

4919 **NAME**

4920 atoi — convert a string to an integer

4921 **SYNOPSIS**

4922 #include <stdlib.h>

4923 int atoi(const char *str);

4924 **DESCRIPTION**

4925 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 4926 conflict between the requirements described here and the ISO C standard is unintentional. This
 4927 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4928 The call *atoi(str)* shall be equivalent to:

4929 (int) strtol(str, (char **)NULL, 10)

4930 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 4931 undefined.

4932 **RETURN VALUE**4933 The *atoi()* function shall return the converted value if the value can be represented.4934 **ERRORS**

4935 No errors are defined.

4936 **EXAMPLES**4937 **Converting an Argument**

4938 The following example checks for proper usage of the program. If there is an argument and the
 4939 decimal conversion of this argument (obtained using *atoi()*) is greater than 0, then the program
 4940 has a valid number of minutes to wait for an event.

```

4941         #include <stdlib.h>
4942         #include <stdio.h>
4943         ...
4944         int minutes_to_event;
4945         ...
4946         if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
4947             fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
4948         }
4949         ...
```

4950 **APPLICATION USAGE**

4951 The *atoi()* function is subsumed by *strtol()* but is retained because it is used extensively in
 4952 existing code. If the number is not known to be in range, *strtol()* should be used because *atoi()* is
 4953 not required to perform any error checking.

4954 **RATIONALE**

4955 None.

4956 **FUTURE DIRECTIONS**

4957 None.

4958 **SEE ALSO**4959 *strtol()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

CHANGE HISTORY

4960
4961 First released in Issue 1. Derived from Issue 1 of the SVID.

4962 NAME

4963 `atol`, `atoll` — convert a string to a long integer

4964 SYNOPSIS

4965 `#include <stdlib.h>`

4966 `long atol(const char *str);`

4967 `long long atoll(const char *nptr);`

4968 DESCRIPTION

4969 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
4970 conflict between the requirements described here and the ISO C standard is unintentional. This
4971 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4972 The call `atol(str)` shall be equivalent to:

4973 `strtoul(str, (char **)NULL, 10)`

4974 The call `atoll(str)` shall be equivalent to:

4975 `strtoll(nptr, (char **)NULL, 10)`

4976 except that the handling of errors may differ. If the value cannot be represented, the behavior is
4977 undefined.

4978 RETURN VALUE

4979 These functions shall return the converted value if the value can be represented.

4980 ERRORS

4981 No errors are defined.

4982 EXAMPLES

4983 None.

4984 APPLICATION USAGE

4985 The `atol()` function is subsumed by `strtoul()` but is retained because it is used extensively in
4986 existing code. If the number is not known to be in range, `strtoul()` should be used because `atol()` is
4987 not required to perform any error checking.

4988 RATIONALE

4989 None.

4990 FUTURE DIRECTIONS

4991 None.

4992 SEE ALSO

4993 `strtoul()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`

4994 CHANGE HISTORY

4995 First released in Issue 1. Derived from Issue 1 of the SVID.

4996 Issue 6

4997 The `atoll()` function is added for alignment with the ISO/IEC 9899:1999 standard.

NAME

basename — return the last component of a pathname

SYNOPSIS

```
#include <libgen.h>

char *basename(char *path);
```

DESCRIPTION

The *basename()* function shall take the pathname pointed to by *path* and return a pointer to the final component of the pathname, deleting any trailing '/' characters.

If the string consists entirely of the '/' character, *basename()* shall return a pointer to the string "/". If the string is exactly "///", it is implementation-defined whether '/' or "///" is returned.

If *path* is a null pointer or points to an empty string, *basename()* shall return a pointer to the string ".".

The *basename()* function may modify the string pointed to by *path*, and may return a pointer to static storage that may then be overwritten by a subsequent call to *basename()*.

The *basename()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

The *basename()* function shall return a pointer to the final component of *path*.

ERRORS

No errors are defined.

EXAMPLES

Using basename()

The following program fragment returns a pointer to the value *lib*, which is the base name of */usr/lib*.

```
#include <libgen.h>
...
char *name = "/usr/lib";
char *base;

base = basename(name);
...
```

Sample Input and Output Strings for basename()

In the following table, the input string is the value pointed to by *path*, and the output string is the return value of the *basename()* function.

Input String	Output String
"/usr/lib"	"lib"
"/usr/"	"usr"
"/"	"/"
"/" / "///"	"/"
"/usr//lib//"	"lib"

5039 **APPLICATION USAGE**

5040 None.

5041 **RATIONALE**

5042 None.

5043 **FUTURE DIRECTIONS**

5044 None.

5045 **SEE ALSO**

5046 *dirname()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<libgen.h>**, the Shell and
5047 Utilities volume of IEEE Std 1003.1-2001, *basename*

5048 **CHANGE HISTORY**

5049 First released in Issue 4, Version 2.

5050 **Issue 5**

5051 Moved from X/OPEN UNIX extension to BASE.

5052 Normative text previously in the APPLICATION USAGE section is moved to the
5053 DESCRIPTION.

5054 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5055 **Issue 6**

5056 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

5057 **NAME**5058 bcmp — memory operations (**LEGACY**)5059 **SYNOPSIS**

5060 xSI #include <strings.h>

5061 int bcmp(const void *s1, const void *s2, size_t n);

5062

5063 **DESCRIPTION**5064 The *bcmp()* function shall compare the first *n* bytes of the area pointed to by *s1* with the area
5065 pointed to by *s2*.5066 **RETURN VALUE**5067 The *bcmp()* function shall return 0 if *s1* and *s2* are identical; otherwise, it shall return non-zero.
5068 Both areas are assumed to be *n* bytes long. If the value of *n* is 0, *bcmp()* shall return 0.5069 **ERRORS**

5070 No errors are defined.

5071 **EXAMPLES**

5072 None.

5073 **APPLICATION USAGE**5074 The *memcmp()* function is preferred over this function.5075 For maximum portability, it is recommended to replace the function call to *bcmp()* as follows:

5076 #define bcmp(b1,b2,len) memcmp((b1), (b2), (size_t)(len))

5077 **RATIONALE**

5078 None.

5079 **FUTURE DIRECTIONS**

5080 This function may be withdrawn in a future version.

5081 **SEE ALSO**5082 *memcmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <strings.h>5083 **CHANGE HISTORY**

5084 First released in Issue 4, Version 2.

5085 **Issue 5**

5086 Moved from X/OPEN UNIX extension to BASE.

5087 **Issue 6**

5088 This function is marked LEGACY.

5089 **NAME**5090 bcopy — memory operations (**LEGACY**)5091 **SYNOPSIS**

5092 xSI #include <strings.h>

5093 void bcopy(const void *s1, void *s2, size_t n);

5094

5095 **DESCRIPTION**5096 The *bcopy()* function shall copy *n* bytes from the area pointed to by *s1* to the area pointed to by
5097 *s2*.5098 The bytes are copied correctly even if the area pointed to by *s1* overlaps the area pointed to by
5099 *s2*.5100 **RETURN VALUE**5101 The *bcopy()* function shall not return a value.5102 **ERRORS**

5103 No errors are defined.

5104 **EXAMPLES**

5105 None.

5106 **APPLICATION USAGE**5107 The *memmove()* function is preferred over this function.

5108 The following are approximately equivalent (note the order of the arguments):

5109 bcopy(s1,s2,n) ~ memmove(s2,s1,n)

5110 For maximum portability, it is recommended to replace the function call to *bcopy()* as follows:

5111 #define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void) 0)

5112 **RATIONALE**

5113 None.

5114 **FUTURE DIRECTIONS**

5115 This function may be withdrawn in a future version.

5116 **SEE ALSO**5117 *memmove()*, the Base Definitions volume of IEEE Std 1003.1-2001, <strings.h>5118 **CHANGE HISTORY**

5119 First released in Issue 4, Version 2.

5120 **Issue 5**

5121 Moved from X/OPEN UNIX extension to BASE.

5122 **Issue 6**

5123 This function is marked LEGACY.

5124 NAME

5125 bind — bind a name to a socket

5126 SYNOPSIS

```
5127       #include <sys/socket.h>

5128       int bind(int socket, const struct sockaddr *address,
5129               socklen_t address_len);
```

5130 DESCRIPTION

5131 The *bind()* function shall assign a local socket address *address* to a socket identified by descriptor
5132 *socket* that has no local socket address assigned. Sockets created with the *socket()* function are
5133 initially unnamed; they are identified only by their address family.

5134 The *bind()* function takes the following arguments:

5135 <i>socket</i>	Specifies the file descriptor of the socket to be bound.
5136 <i>address</i>	Points to a sockaddr structure containing the address to be bound to the 5137 socket. The length and format of the address depend on the address family of 5138 the socket.
5139 <i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> 5140 argument.

5141 The socket specified by *socket* may require the process to have appropriate privileges to use the
5142 *bind()* function.

5143 RETURN VALUE

5144 Upon successful completion, *bind()* shall return 0; otherwise, *-1* shall be returned and *errno* set
5145 to indicate the error.

5146 ERRORS

5147 The *bind()* function shall fail if:

5148 [EADDRINUSE]	The specified address is already in use.
5149 [EADDRNOTAVAIL]	The specified address is not available from the local machine.
5150 [EAFNOSUPPORT]	The specified address is not a valid address for the address family of the 5151 specified socket.
5152 [EBADF]	The <i>socket</i> argument is not a valid file descriptor.
5153 [EINVAL]	The socket is already bound to an address, and the protocol does not support 5154 binding to a new address; or the socket has been shut down.
5155 [ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
5156 [EOPNOTSUPP]	The socket type of the specified socket does not support binding to an 5157 address.
5158 If the address family of the socket is AF_UNIX, then <i>bind()</i> shall fail if:	
5159 [EACCES]	A component of the path prefix denies search permission, or the requested 5160 name requires writing in a directory with a mode that denies write 5161 permission.
5162 [EDESTADDRREQ] or [EISDIR]	The <i>address</i> argument is a null pointer.

5166	[EIO]	An I/O error occurred.
5167	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname
5168		in <i>address</i> .
5169	[ENAMETOOLONG]	
5170		A component of a pathname exceeded {NAME_MAX} characters, or an entire
5171		pathname exceeded {PATH_MAX} characters.
5172	[ENOENT]	A component of the pathname does not name an existing file or the pathname
5173		is an empty string.
5174	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> is not a directory.
5175	[EROFS]	The name would reside on a read-only file system.
5176		The <i>bind()</i> function may fail if:
5177	[EACCES]	The specified address is protected and the current user does not have
5178		permission to bind to it.
5179	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family.
5180	[EISCONN]	The socket is already connected.
5181	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
5182		resolution of the pathname in <i>address</i> .
5183	[ENAMETOOLONG]	
5184		Pathname resolution of a symbolic link produced an intermediate result
5185		whose length exceeds {PATH_MAX}.
5186	[ENOBUFS]	Insufficient resources were available to complete the call.

5187 EXAMPLES

5188 None.

5189 APPLICATION USAGE

5190 An application program can retrieve the assigned socket name with the *getsockname()* function.

5191 RATIONALE

5192 None.

5193 FUTURE DIRECTIONS

5194 None.

5195 SEE ALSO

5196 *connect()*, *getsockname()*, *listen()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,
5197 <**sys/socket.h**>

5198 CHANGE HISTORY

5199 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

5200 **NAME**

5201 bsd_signal — simplified signal facilities

5202 **SYNOPSIS**

5203 OB XSI #include <signal.h>

5204 void (*bsd_signal(int sig, void (*func)(int)))(int);

5205

5206 **DESCRIPTION**5207 The *bsd_signal()* function provides a partially compatible interface for programs written to
5208 historical system interfaces (see APPLICATION USAGE).5209 The function call *bsd_signal(sig, func)* shall be equivalent to the following:

```

5210 void (*bsd_signal(int sig, void (*func)(int)))(int)
5211 {
5212     struct sigaction act, oact;
5213
5214     act.sa_handler = func;
5215     act.sa_flags = SA_RESTART;
5216     sigemptyset(&act.sa_mask);
5217     sigaddset(&act.sa_mask, sig);
5218     if (sigaction(sig, &act, &oact) == -1)
5219         return(SIG_ERR);
5219     return(oact.sa_handler);
5220 }

```

5221 The handler function should be declared:

5222 void handler(int sig);

5223 where *sig* is the signal number. The behavior is undefined if *func* is a function that takes more
5224 than one argument, or an argument of a different type.5225 **RETURN VALUE**5226 Upon successful completion, *bsd_signal()* shall return the previous action for *sig*. Otherwise,
5227 SIG_ERR shall be returned and *errno* shall be set to indicate the error.5228 **ERRORS**5229 Refer to *sigaction()*.5230 **EXAMPLES**

5231 None.

5232 **APPLICATION USAGE**

5233 This function is a direct replacement for the BSD *signal()* function for simple applications that
5234 are installing a single-argument signal handler function. If a BSD signal handler function is being
5235 installed that expects more than one argument, the application has to be modified to use
5236 *sigaction()*. The *bsd_signal()* function differs from *signal()* in that the SA_RESTART flag is set
5237 and the SA_RESETHAND is clear when *bsd_signal()* is used. The state of these flags is not
5238 specified for *signal()*.

5239 It is recommended that new applications use the *sigaction()* function.5240 **RATIONALE**

5241 None.

5242 **FUTURE DIRECTIONS**

5243 None.

5244 **SEE ALSO**

5245 *sigaction()*, *sigaddset()*, *sigemptyset()*, *signal()*, the Base Definitions volume of
5246 IEEE Std 1003.1-2001, <**signal.h**>

5247 **CHANGE HISTORY**

5248 First released in Issue 4, Version 2.

5249 **Issue 5**

5250 Moved from X/OPEN UNIX extension to BASE.

5251 **Issue 6**

5252 This function is marked obsolescent.

5253 **NAME**

5254 bsearch — binary search a sorted table

5255 **SYNOPSIS**

5256 #include <stdlib.h>

```
5257       void *bsearch(const void *key, const void *base, size_t nel,
5258                   size_t width, int (*compar)(const void *, const void *));
```

5259 **DESCRIPTION**

5260 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 5261 conflict between the requirements described here and the ISO C standard is unintentional. This
 5262 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5263 The *bsearch()* function shall search an array of *nel* objects, the initial element of which is pointed
 5264 to by *base*, for an element that matches the object pointed to by *key*. The size of each element in
 5265 the array is specified by *width*. If the *nel* argument has the value zero, the comparison function
 5266 pointed to by *compar* shall not be called and no match shall be found.

5267 The comparison function pointed to by *compar* shall be called with two arguments that point to
 5268 the *key* object and to an array element, in that order.

5269 The application shall ensure that the comparison function pointed to by *compar* does not alter the
 5270 contents of the array. The implementation may reorder elements of the array between calls to the
 5271 comparison function, but shall not alter the contents of any individual element.

5272 The implementation shall ensure that the first argument is always a pointer to the key.

5273 When the same objects (consisting of *width* bytes, irrespective of their current positions in the
 5274 array) are passed more than once to the comparison function, the results shall be consistent with
 5275 one another. That is, the same object shall always compare the same way with the key.

5276 The application shall ensure that the function returns an integer less than, equal to, or greater
 5277 than 0 if the *key* object is considered, respectively, to be less than, to match, or to be greater than
 5278 the array element. The application shall ensure that the array consists of all the elements that
 5279 compare less than, all the elements that compare equal to, and all the elements that compare
 5280 greater than the *key* object, in that order.

5281 **RETURN VALUE**

5282 The *bsearch()* function shall return a pointer to a matching member of the array, or a null pointer
 5283 if no match is found. If two or more members compare equal, which member is returned is
 5284 unspecified.

5285 **ERRORS**

5286 No errors are defined.

5287 **EXAMPLES**

5288 The example below searches a table containing pointers to nodes consisting of a string and its
 5289 length. The table is ordered alphabetically on the string in the node pointed to by each entry.

5290 The code fragment below reads in strings and either finds the corresponding node and prints out
 5291 the string and its length, or prints an error message.

```
5292     #include <stdio.h>
5293     #include <stdlib.h>
5294     #include <string.h>

5295     #define TABSIZE     1000
```

```

5296     struct node {                                /* These are stored in the table. */
5297         char *string;
5298         int length;
5299     };
5300     struct node table[TABSIZE];    /* Table to be searched. */
5301     .
5302     .
5303     .
5304     {
5305         struct node *node_ptr, node;
5306         /* Routine to compare 2 nodes. */
5307         int node_compare(const void *, const void *);
5308         char str_space[20];    /* Space to read string into. */
5309         .
5310         .
5311         .
5312         node.string = str_space;
5313         while (scanf("%s", node.string) != EOF) {
5314             node_ptr = (struct node *)bsearch((void *)(&node),
5315                 (void *)table, TABSIZE,
5316                 sizeof(struct node), node_compare);
5317             if (node_ptr != NULL) {
5318                 (void)printf("string = %20s, length = %d\n",
5319                     node_ptr->string, node_ptr->length);
5320             } else {
5321                 (void)printf("not found: %s\n", node.string);
5322             }
5323         }
5324     }
5325     /*
5326         This routine compares two nodes based on an
5327         alphabetical ordering of the string field.
5328     */
5329     int
5330     node_compare(const void *node1, const void *node2)
5331     {
5332         return strcmp(((const struct node *)node1)->string,
5333             ((const struct node *)node2)->string);
5334     }

```

5335 APPLICATION USAGE

5336 The pointers to the key and the element at the base of the table should be of type pointer-to-
5337 element.

5338 The comparison function need not compare every byte, so arbitrary data may be contained in
5339 the elements in addition to the values being compared.

5340 In practice, the array is usually sorted according to the comparison function.

5341 RATIONALE

5342 The requirement that the second argument (hereafter referred to as *p*) to the comparison
5343 function is a pointer to an element of the array implies that for every call all of the following
5344 expressions are non-zero: |

```
5345      ((char *)p - (char *(base) % width == 0
5346      (char *)p >= (char *)base
5347      (char *)p < (char *)base + nel * width
```

5348 FUTURE DIRECTIONS

5349 None.

5350 SEE ALSO

5351 *hcreate()*, *lsearch()*, *qsort()*, *tsearch()*, the Base Definitions volume of IEEE Std 1003.1-2001,
5352 **<stdlib.h>**

5353 CHANGE HISTORY

5354 First released in Issue 1. Derived from Issue 1 of the SVID.

5355 Issue 6

5356 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

5357 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the
5358 DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three
5359 paragraphs. The RATIONALE section is also updated. These changes are for alignment with the
5360 ISO C standard.

5361 **NAME**

5362 btowc — single byte to wide character conversion

5363 **SYNOPSIS**

5364 #include <stdio.h>

5365 #include <wchar.h>

5366 wint_t btowc(int c);

5367 **DESCRIPTION**

5368 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5369 conflict between the requirements described here and the ISO C standard is unintentional. This
5370 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5371 The *btowc()* function shall determine whether *c* constitutes a valid (one-byte) character in the
5372 initial shift state.

5373 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

5374 **RETURN VALUE**

5375 The *btowc()* function shall return WEOF if *c* has the value EOF or if (**unsigned char**) *c* does not
5376 constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the
5377 wide-character representation of that character.

5378 **ERRORS**

5379 No errors are defined.

5380 **EXAMPLES**

5381 None.

5382 **APPLICATION USAGE**

5383 None.

5384 **RATIONALE**

5385 None.

5386 **FUTURE DIRECTIONS**

5387 None.

5388 **SEE ALSO**5389 *wctob()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>5390 **CHANGE HISTORY**

5391 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
5392 (E).

5393 **NAME**5394 bzero — memory operations (**LEGACY**)5395 **SYNOPSIS**

5396 xSI #include <strings.h>

5397 void bzero(void *s, size_t n);

5398

5399 **DESCRIPTION**5400 The *bzero()* function shall place *n* zero-valued bytes in the area pointed to by *s*.5401 **RETURN VALUE**5402 The *bzero()* function shall not return a value.5403 **ERRORS**

5404 No errors are defined.

5405 **EXAMPLES**

5406 None.

5407 **APPLICATION USAGE**5408 The *memset()* function is preferred over this function.5409 For maximum portability, it is recommended to replace the function call to *bzero()* as follows:

5410 #define bzero(b,len) (memset((b), '\0', (len)), (void) 0)

5411 **RATIONALE**

5412 None.

5413 **FUTURE DIRECTIONS**

5414 This function may be withdrawn in a future version.

5415 **SEE ALSO**5416 *memset()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**strings.h**>5417 **CHANGE HISTORY**

5418 First released in Issue 4, Version 2.

5419 **Issue 5**

5420 Moved from X/OPEN UNIX extension to BASE.

5421 **Issue 6**

5422 This function is marked LEGACY.

5423 **NAME**

5424 cabs, cabsf, cabsl — return a complex absolute value

5425 **SYNOPSIS**

5426 #include <complex.h>

5427 double cabs(double complex z);

5428 float cabsf(float complex z);

5429 long double cabsl(long double complex z);

5430 **DESCRIPTION**

5431 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5432 conflict between the requirements described here and the ISO C standard is unintentional. This
5433 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5434 These functions shall compute the complex absolute value (also called norm, modulus, or
5435 magnitude) of z.

5436 **RETURN VALUE**

5437 These functions shall return the complex absolute value.

5438 **ERRORS**

5439 No errors are defined.

5440 **EXAMPLES**

5441 None.

5442 **APPLICATION USAGE**

5443 None.

5444 **RATIONALE**

5445 None.

5446 **FUTURE DIRECTIONS**

5447 None.

5448 **SEE ALSO**

5449 The Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>

5450 **CHANGE HISTORY**

5451 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5452 **NAME**

5453 cacos, cacosf, cacosl — complex arc cosine functions

5454 **SYNOPSIS**

5455 #include <complex.h>

5456 double complex cacos(double complex z);

5457 float complex cacosf(float complex z);

5458 long double complex cacosl(long double complex z);

5459 **DESCRIPTION**

5460 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5461 conflict between the requirements described here and the ISO C standard is unintentional. This
5462 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5463 These functions shall compute the complex arc cosine of z , with branch cuts outside the interval
5464 $[-1, +1]$ along the real axis.

5465 **RETURN VALUE**

5466 These functions shall return the complex arc cosine value, in the range of a strip mathematically
5467 unbounded along the imaginary axis and in the interval $[0, \pi]$ along the real axis.

5468 **ERRORS**

5469 No errors are defined.

5470 **EXAMPLES**

5471 None.

5472 **APPLICATION USAGE**

5473 None.

5474 **RATIONALE**

5475 None.

5476 **FUTURE DIRECTIONS**

5477 None.

5478 **SEE ALSO**5479 `ccos()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**5480 **CHANGE HISTORY**

5481 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5482 **NAME**

5483 cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

5484 **SYNOPSIS**

5485 #include <complex.h>

5486 double complex cacosh(double complex z);

5487 float complex cacoshf(float complex z);

5488 long double complex cacoshl(long double complex z);

5489 **DESCRIPTION**

5490 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5491 conflict between the requirements described here and the ISO C standard is unintentional. This
5492 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5493 These functions shall compute the complex arc hyperbolic cosine of z , with a branch cut at
5494 values less than 1 along the real axis.

5495 **RETURN VALUE**

5496 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip
5497 of non-negative values along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

5498 **ERRORS**

5499 No errors are defined.

5500 **EXAMPLES**

5501 None.

5502 **APPLICATION USAGE**

5503 None.

5504 **RATIONALE**

5505 None.

5506 **FUTURE DIRECTIONS**

5507 None.

5508 **SEE ALSO**

5509 ccosh(), the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>

5510 **CHANGE HISTORY**

5511 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5512 **NAME**

5513 cacosl — complex arc cosine functions

5514 **SYNOPSIS**

5515 #include <complex.h>

5516 long double complex cacosl(long double complex *z*);5517 **DESCRIPTION**5518 Refer to *cacos*(.).

5519 **NAME**

5520 calloc — a memory allocator

5521 **SYNOPSIS**

5522 #include <stdlib.h>

5523 void *calloc(size_t *nelem*, size_t *elsize*);5524 **DESCRIPTION**

5525 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 5526 conflict between the requirements described here and the ISO C standard is unintentional. This
 5527 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5528 The *calloc()* function shall allocate unused space for an array of *nelem* elements each of whose
 5529 size in bytes is *elsize*. The space shall be initialized to all bits 0.

5530 The order and contiguity of storage allocated by successive calls to *calloc()* is unspecified. The
 5531 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 5532 a pointer to any type of object and then used to access such an object or an array of such objects
 5533 in the space allocated (until the space is explicitly freed or reallocated). Each such allocation
 5534 shall yield a pointer to an object disjoint from any other object. The pointer returned shall point
 5535 to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null
 5536 pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-
 5537 defined: the value returned shall be either a null pointer or a unique pointer.

5538 **RETURN VALUE**

5539 Upon successful completion with both *nelem* and *elsize* non-zero, *calloc()* shall return a pointer to
 5540 the allocated space. If either *nelem* or *elsize* is 0, then either a null pointer or a unique pointer
 5541 value that can be successfully passed to *free()* shall be returned. Otherwise, it shall return a null
 5542 CX pointer and set *errno* to indicate the error.

5543 **ERRORS**5544 The *calloc()* function shall fail if:

5545 CX [ENOMEM] Insufficient memory is available.

5546 **EXAMPLES**

5547 None.

5548 **APPLICATION USAGE**

5549 There is now no requirement for the implementation to support the inclusion of <malloc.h>.

5550 **RATIONALE**

5551 None.

5552 **FUTURE DIRECTIONS**

5553 None.

5554 **SEE ALSO**5555 *free()*, *malloc()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>5556 **CHANGE HISTORY**

5557 First released in Issue 1. Derived from Issue 1 of the SVID.

5558 **Issue 6**

5559 Extensions beyond the ISO C standard are marked.

- 5560 The following new requirements on POSIX implementations derive from alignment with the
5561 Single UNIX Specification:
- 5562 • The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient
5563 memory condition occurs.

5564 **NAME**

5565 carg, cargf, cargl — complex argument functions

5566 **SYNOPSIS**

5567 #include <complex.h>

5568 double carg(double complex z);

5569 float cargf(float complex z);

5570 long double cargl(long double complex z);

5571 **DESCRIPTION**

5572 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5573 conflict between the requirements described here and the ISO C standard is unintentional. This
5574 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5575 These functions shall compute the argument (also called phase angle) of z , with a branch cut
5576 along the negative real axis.

5577 **RETURN VALUE**5578 These functions shall return the value of the argument in the interval $[-\pi, +\pi]$.5579 **ERRORS**

5580 No errors are defined.

5581 **EXAMPLES**

5582 None.

5583 **APPLICATION USAGE**

5584 None.

5585 **RATIONALE**

5586 None.

5587 **FUTURE DIRECTIONS**

5588 None.

5589 **SEE ALSO**5590 *cimag()*, *conj()*, *cproj()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>5591 **CHANGE HISTORY**

5592 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5593 NAME

5594 casin, casinf, casinl — complex arc sine functions

5595 SYNOPSIS

5596 #include <complex.h>

5597 double complex casin(double complex z);

5598 float complex casinf(float complex z);

5599 long double complex casinl(long double complex z);

5600 DESCRIPTION

5601 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5602 conflict between the requirements described here and the ISO C standard is unintentional. This
5603 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5604 These functions shall compute the complex arc sine of z , with branch cuts outside the interval
5605 $[-1, +1]$ along the real axis.

5606 RETURN VALUE

5607 These functions shall return the complex arc sine value, in the range of a strip mathematically
5608 unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the real axis.

5609 ERRORS

5610 No errors are defined.

5611 EXAMPLES

5612 None.

5613 APPLICATION USAGE

5614 None.

5615 RATIONALE

5616 None.

5617 FUTURE DIRECTIONS

5618 None.

5619 SEE ALSO

5620 *csin()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>

5621 CHANGE HISTORY

5622 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5623 **NAME**

5624 casinh, casinhf, casinhl — complex arc hyperbolic sine functions

5625 **SYNOPSIS**

5626 #include <complex.h>

5627 double complex casinh(double complex z);

5628 float complex casinhf(float complex z);

5629 long double complex casinhl(long double complex z);

5630 **DESCRIPTION**

5631 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5632 conflict between the requirements described here and the ISO C standard is unintentional. This
5633 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5634 These functions shall compute the complex arc hyperbolic sine of z , with branch cuts outside the
5635 interval $[-i, +i]$ along the imaginary axis.

5636 **RETURN VALUE**

5637 These functions shall return the complex arc hyperbolic sine value, in the range of a strip
5638 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
5639 imaginary axis.

5640 **ERRORS**

5641 No errors are defined.

5642 **EXAMPLES**

5643 None.

5644 **APPLICATION USAGE**

5645 None.

5646 **RATIONALE**

5647 None.

5648 **FUTURE DIRECTIONS**

5649 None.

5650 **SEE ALSO**5651 `csinh()`, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>5652 **CHANGE HISTORY**

5653 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5654 **NAME**5655 `casinl` — complex arc sine functions5656 **SYNOPSIS**5657 `#include <complex.h>`5658 `long double complex casinl(long double complex z);`5659 **DESCRIPTION**5660 Refer to *casin()*.

5661 **NAME**

5662 catan, catanf, catanl — complex arc tangent functions

5663 **SYNOPSIS**

5664 #include <complex.h>

5665 double complex catan(double complex z);

5666 float complex catanf(float complex z);

5667 long double complex catanl(long double complex z);

5668 **DESCRIPTION**

5669 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5670 conflict between the requirements described here and the ISO C standard is unintentional. This
5671 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5672 These functions shall compute the complex arc tangent of z , with branch cuts outside the
5673 interval $[-i, +i]$ along the imaginary axis.

5674 **RETURN VALUE**

5675 These functions shall return the complex arc tangent value, in the range of a strip
5676 mathematically unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the
5677 real axis.

5678 **ERRORS**

5679 No errors are defined.

5680 **EXAMPLES**

5681 None.

5682 **APPLICATION USAGE**

5683 None.

5684 **RATIONALE**

5685 None.

5686 **FUTURE DIRECTIONS**

5687 None.

5688 **SEE ALSO**

5689 ctan(), the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>

5690 **CHANGE HISTORY**

5691 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5692 **NAME**

5693 catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

5694 **SYNOPSIS**

5695 #include <complex.h>

5696 double complex catanh(double complex z);

5697 float complex catanhf(float complex z);

5698 long double complex catanhl(long double complex z);

5699 **DESCRIPTION**

5700 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5701 conflict between the requirements described here and the ISO C standard is unintentional. This
5702 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5703 These functions shall compute the complex arc hyperbolic tangent of z , with branch cuts outside
5704 the interval $[-1, +1]$ along the real axis.

5705 **RETURN VALUE**

5706 These functions shall return the complex arc hyperbolic tangent value, in the range of a strip
5707 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
5708 imaginary axis.

5709 **ERRORS**

5710 No errors are defined.

5711 **EXAMPLES**

5712 None.

5713 **APPLICATION USAGE**

5714 None.

5715 **RATIONALE**

5716 None.

5717 **FUTURE DIRECTIONS**

5718 None.

5719 **SEE ALSO**5720 `ctanh()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**5721 **CHANGE HISTORY**

5722 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5723 **NAME**

5724 catanl — complex arc tangent functions

5725 **SYNOPSIS**

5726 #include <complex.h>

5727 long double complex catanl(long double complex *z*);5728 **DESCRIPTION**5729 Refer to *catan()*.

5730 **NAME**

5731 catclose — close a message catalog descriptor

5732 **SYNOPSIS**

5733 xSI #include <nl_types.h>

5734 int catclose(nl_catd catd);

5735

5736 **DESCRIPTION**

5737 The *catclose()* function shall close the message catalog identified by *catd*. If a file descriptor is
5738 used to implement the type **nl_catd**, that file descriptor shall be closed.

5739 **RETURN VALUE**

5740 Upon successful completion, *catclose()* shall return 0; otherwise, *-1* shall be returned, and *errno*
5741 set to indicate the error.

5742 **ERRORS**5743 The *catclose()* function may fail if:

5744 [EBADF] The catalog descriptor is not valid.

5745 [EINTR] The *catclose()* function was interrupted by a signal.5746 **EXAMPLES**

5747 None.

5748 **APPLICATION USAGE**

5749 None.

5750 **RATIONALE**

5751 None.

5752 **FUTURE DIRECTIONS**

5753 None.

5754 **SEE ALSO**5755 *catgets()*, *catopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**nl_types.h**>5756 **CHANGE HISTORY**

5757 First released in Issue 2.

5758 **NAME**

5759 catgets — read a program message

5760 **SYNOPSIS**

5761 xSI #include <nl_types.h>

5762 char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);

5763

5764 **DESCRIPTION**

5765 The *catgets()* function shall attempt to read message *msg_id*, in set *set_id*, from the message catalog identified by *catd*. The *catd* argument is a message catalog descriptor returned from an earlier call to *catopen()*. The *s* argument points to a default message string which shall be returned by *catgets()* if it cannot retrieve the identified message.

5769 The *catgets()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

5771 **RETURN VALUE**

5772 If the identified message is retrieved successfully, *catgets()* shall return a pointer to an internal buffer area containing the null-terminated message string. If the call is unsuccessful for any reason, *s* shall be returned and *errno* may be set to indicate the error.

5775 **ERRORS**5776 The *catgets()* function may fail if:

5777 [EBADF] The *catd* argument is not a valid message catalog descriptor open for reading.

5778 [EBADMSG] The message identified by *set_id* and *msg_id* in the specified message catalog did not satisfy implementation-defined security criteria.

5780 [EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

5782 [EINVAL] The message catalog identified by *catd* is corrupted.

5783 [ENOMSG] The message identified by *set_id* and *msg_id* is not in the message catalog.

5784 **EXAMPLES**

5785 None.

5786 **APPLICATION USAGE**

5787 None.

5788 **RATIONALE**

5789 None.

5790 **FUTURE DIRECTIONS**

5791 None.

5792 **SEE ALSO**5793 *catclose()*, *catopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <nl_types.h>5794 **CHANGE HISTORY**

5795 First released in Issue 2.

5796 **Issue 5**

5797 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5798 **Issue 6**

5799 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

5800 **NAME**

5801 catopen — open a message catalog

5802 **SYNOPSIS**

5803 xSI #include <nl_types.h>

5804 nl_catd catopen(const char *name, int oflag);

5805

5806 **DESCRIPTION**

5807 The *catopen()* function shall open a message catalog and return a message catalog descriptor.
 5808 The *name* argument specifies the name of the message catalog to be opened. If *name* contains a
 5809 ' / ', then *name* specifies a complete name for the message catalog. Otherwise, the environment
 5810 variable *NLSPATH* is used with *name* substituted for the %N conversion specification (see the
 5811 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables). If
 5812 *NLSPATH* exists in the environment when the process starts, then if the process has appropriate
 5813 privileges, the behavior of *catopen()* is undefined. If *NLSPATH* does not exist in the environment,
 5814 or if a message catalog cannot be found in any of the components specified by *NLSPATH*, then
 5815 an implementation-defined default path shall be used. This default may be affected by the
 5816 setting of *LC_MESSAGES* if the value of *oflag* is *NL_CAT_LOCALE*, or the *LANG* environment
 5817 variable if *oflag* is 0.

5818 A message catalog descriptor shall remain valid in a process until that process closes it, or a
 5819 successful call to one of the *exec* functions. A change in the setting of the *LC_MESSAGES*
 5820 category may invalidate existing open catalogs.

5821 If a file descriptor is used to implement message catalog descriptors, the *FD_CLOEXEC* flag
 5822 shall be set; see <fcntl.h>.

5823 If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the
 5824 catalog without regard to the *LC_MESSAGES* category. If the *oflag* argument is
 5825 *NL_CAT_LOCALE*, the *LC_MESSAGES* category is used to locate the message catalog (see the
 5826 Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables).

5827 **RETURN VALUE**

5828 Upon successful completion, *catopen()* shall return a message catalog descriptor for use on
 5829 subsequent calls to *catgets()* and *catclose()*. Otherwise, *catopen()* shall return (**nl_catd**) -1 and set
 5830 *errno* to indicate the error.

5831 **ERRORS**5832 The *catopen()* function may fail if:

5833 [EACCES] Search permission is denied for the component of the path prefix of the
 5834 message catalog or read permission is denied for the message catalog.

5835 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

5836 [ENAMETOOLONG]

5837 The length of a pathname of the message catalog exceeds {PATH_MAX} or a
 5838 pathname component is longer than {NAME_MAX}.

5839 [ENAMETOOLONG]

5840 Pathname resolution of a symbolic link produced an intermediate result
 5841 whose length exceeds {PATH_MAX}.

5842 [ENFILE] Too many files are currently open in the system.

5843 [ENOENT] The message catalog does not exist or the *name* argument points to an empty
 5844 string.

5845 [ENOMEM] Insufficient storage space is available.

5846 [ENOTDIR] A component of the path prefix of the message catalog is not a directory.

5847 **EXAMPLES**

5848 None.

5849 **APPLICATION USAGE**

5850 Some implementations of *catopen()* use *malloc()* to allocate space for internal buffer areas. The
5851 *catopen()* function may fail if there is insufficient storage space available to accommodate these
5852 buffers.

5853 Conforming applications must assume that message catalog descriptors are not valid after a call
5854 to one of the *exec* functions.

5855 Application writers should be aware that guidelines for the location of message catalogs have
5856 not yet been developed. Therefore they should take care to avoid conflicting with catalogs used
5857 by other applications and the standard utilities.

5858 **RATIONALE**

5859 None.

5860 **FUTURE DIRECTIONS**

5861 None.

5862 **SEE ALSO**

5863 *catclose()*, *catgets()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**,
5864 **<nl_types.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001

5865 **CHANGE HISTORY**

5866 First released in Issue 2.

5867 **NAME**

5868 cbrt, cbrtf, cbrtl — cube root functions

5869 **SYNOPSIS**

5870 #include <math.h>

5871 double cbrt(double x);

5872 float cbrtf(float x);

5873 long double cbrtl(long double x);

5874 **DESCRIPTION**

5875 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 5876 conflict between the requirements described here and the ISO C standard is unintentional. This
 5877 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5878 These functions shall compute the real cube root of their argument *x*.5879 **RETURN VALUE**5880 Upon successful completion, these functions shall return the cube root of *x*.5881 **MX** If *x* is NaN, a NaN shall be returned.5882 If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.5883 **ERRORS**

5884 No errors are defined.

5885 **EXAMPLES**

5886 None.

5887 **APPLICATION USAGE**

5888 None.

5889 **RATIONALE**

5890 For some applications, a true cube root function, which returns negative results for negative
 5891 arguments, is more appropriate than *pow(x, 1.0/3.0)*, which returns a NaN for *x* less than 0.

5892 **FUTURE DIRECTIONS**

5893 None.

5894 **SEE ALSO**

5895 The Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

5896 **CHANGE HISTORY**

5897 First released in Issue 4, Version 2.

5898 **Issue 5**

5899 Moved from X/OPEN UNIX extension to BASE.

5900 **Issue 6**5901 The *cbrt()* function is no longer marked as an extension.5902 The *cbrtf()* and *cbrtl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

5903 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 5904 revised to align with the ISO/IEC 9899:1999 standard.

5905 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 5906 marked.

5907 NAME

5908 `ccos, ccosh, ccoshl` — complex cosine functions

5909 SYNOPSIS

5910 `#include <complex.h>`

5911 `double complex ccos(double complex z);`

5912 `float complex ccosh(float complex z);`

5913 `long double complex ccoshl(long double complex z);`

5914 DESCRIPTION

5915 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5916 conflict between the requirements described here and the ISO C standard is unintentional. This
5917 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5918 These functions shall compute the complex cosine of *z*.

5919 RETURN VALUE

5920 These functions shall return the complex cosine value.

5921 ERRORS

5922 No errors are defined.

5923 EXAMPLES

5924 None.

5925 APPLICATION USAGE

5926 None.

5927 RATIONALE

5928 None.

5929 FUTURE DIRECTIONS

5930 None.

5931 SEE ALSO

5932 `cacos()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<complex.h>**

5933 CHANGE HISTORY

5934 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5935 **NAME**

5936 ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

5937 **SYNOPSIS**

5938 #include <complex.h>

5939 double complex ccosh(double complex z);

5940 float complex ccoshf(float complex z);

5941 long double complex ccoshl(long double complex z);

5942 **DESCRIPTION**

5943 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5944 conflict between the requirements described here and the ISO C standard is unintentional. This
5945 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5946 These functions shall compute the complex hyperbolic cosine of *z*.5947 **RETURN VALUE**

5948 These functions shall return the complex hyperbolic cosine value.

5949 **ERRORS**

5950 No errors are defined.

5951 **EXAMPLES**

5952 None.

5953 **APPLICATION USAGE**

5954 None.

5955 **RATIONALE**

5956 None.

5957 **FUTURE DIRECTIONS**

5958 None.

5959 **SEE ALSO**5960 *cacosh()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>5961 **CHANGE HISTORY**

5962 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5963 **NAME**

5964 ccosl — complex cosine functions

5965 **SYNOPSIS**

5966 #include <complex.h>

5967 long double complex ccosl(long double complex z);

5968 **DESCRIPTION**5969 Refer to *ccos()*.

5970 **NAME**

5971 ceil, ceilf, ceill — ceiling value function

5972 **SYNOPSIS**

5973 #include <math.h>

5974 double ceil(double x);

5975 float ceilf(float x);

5976 long double ceill(long double x);

5977 **DESCRIPTION**

5978 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 5979 conflict between the requirements described here and the ISO C standard is unintentional. This
 5980 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5981 These functions shall compute the smallest integral value not less than *x*.

5982 An application wishing to check for error situations should set *errno* to zero and call
 5983 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 5984 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 5985 zero, an error has occurred.

5986 **RETURN VALUE**

5987 Upon successful completion, *ceil()*, *ceilf()*, and *ceill()* shall return the smallest integral value not
 5988 less than *x*, expressed as a type **double**, **float**, or **long double**, respectively.

5989 MX If *x* is NaN, a NaN shall be returned.5990 If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.

5991 XSI If the correct value would cause overflow, a range error shall occur and *ceil()*, *ceilf()*, and *ceill()*
 5992 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

5993 **ERRORS**

5994 These functions shall fail if:

5995 XSI Range Error The result overflows.

5996 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 5997 then *errno* shall be set to [ERANGE]. If the integer expression
 5998 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 5999 floating-point exception shall be raised.

6000 **EXAMPLES**

6001 None.

6002 **APPLICATION USAGE**

6003 The integral value returned by these functions need not be expressible as an **int** or **long**. The
 6004 return value should be tested before assigning it to an integer type to avoid the undefined results
 6005 of an integer overflow.

6006 The *ceil()* function can only overflow when the floating-point representation has
 6007 DBL_MANT_DIG > DBL_MAX_EXP.

6008 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 6009 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

6010 **RATIONALE**

6011 None.

6012 **FUTURE DIRECTIONS**

6013 None.

6014 **SEE ALSO**

6015 *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
6016 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

6017 **CHANGE HISTORY**

6018 First released in Issue 1. Derived from Issue 1 of the SVID.

6019 **Issue 5**

6020 The DESCRIPTION is updated to indicate how an application should check for an error. This
6021 text was previously published in the APPLICATION USAGE section.

6022 **Issue 6**6023 The *ceilf()* and *ceil()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

6024 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
6025 revised to align with the ISO/IEC 9899:1999 standard.

6026 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
6027 marked.

6028 **NAME**

6029 cexp, cexpf, cexpl — complex exponential functions

6030 **SYNOPSIS**

6031 #include <complex.h>

6032 double complex cexp(double complex z);

6033 float complex cexpf(float complex z);

6034 long double complex cexpl(long double complex z);

6035 **DESCRIPTION**

6036 cx The functionality described on this reference page is aligned with the ISO C standard. Any
6037 conflict between the requirements described here and the ISO C standard is unintentional. This
6038 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6039 These functions shall compute the complex exponent of z , defined as e^z .6040 **RETURN VALUE**6041 These functions shall return the complex exponential value of z .6042 **ERRORS**

6043 No errors are defined.

6044 **EXAMPLES**

6045 None.

6046 **APPLICATION USAGE**

6047 None.

6048 **RATIONALE**

6049 None.

6050 **FUTURE DIRECTIONS**

6051 None.

6052 **SEE ALSO**6053 *clog()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>6054 **CHANGE HISTORY**

6055 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6056 **NAME**

6057 cfgetispeed — get input baud rate

6058 **SYNOPSIS**

6059 #include <termios.h>

6060 speed_t cfgetispeed(const struct termios *termios_p);

6061 **DESCRIPTION**

6062 The *cfgetispeed()* function shall extract the input baud rate from the **termios** structure to which
 6063 the *termios_p* argument points.

6064 This function shall return exactly the value in the **termios** data structure, without interpretation.

6065 **RETURN VALUE**

6066 Upon successful completion, *cfgetispeed()* shall return a value of type **speed_t** representing the
 6067 input baud rate.

6068 **ERRORS**

6069 No errors are defined.

6070 **EXAMPLES**

6071 None.

6072 **APPLICATION USAGE**

6073 None.

6074 **RATIONALE**

6075 The term “baud” is used historically here, but is not technically correct. This is properly “bits
 6076 per second”, which may not be the same as baud. However, the term is used because of the
 6077 historical usage and understanding.

6078 The *cfgetospeed()*, *cfgetispeed()*, *cfsetospeed()*, and *cfsetispeed()* functions do not take arguments as
 6079 numbers, but rather as symbolic names. There are two reasons for this:

- 6080 1. Historically, numbers were not used because of the way the rate was stored in the data
 6081 structure. This is retained even though a function is now used.
- 6082 2. More importantly, only a limited set of possible rates is at all portable, and this constrains
 6083 the application to that set.

6084 There is nothing to prevent an implementation accepting as an extension a number (such as 126),
 6085 and since the encoding of the Bxxx symbols is not specified, this can be done to avoid
 6086 introducing ambiguity.

6087 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications
 6088 in this volume of IEEE Std 1003.1-2001 have made it possible to determine whether split rates are
 6089 supported and to support them without having to treat zero as a special case. Since this
 6090 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is
 6091 the literal constant 0, not the symbolic constant B0. This volume of IEEE Std 1003.1-2001 does not
 6092 preclude B0 from being defined as the value 0; in fact, implementations would likely benefit
 6093 from the two being equivalent. This volume of IEEE Std 1003.1-2001 does not fully specify
 6094 whether the previous *cfsetispeed()* value is retained after a *tcgetattr()* as the actual value or as
 6095 zero. Therefore, conforming applications should always set both the input speed and output
 6096 speed when setting either.

6097 In historical implementations, the baud rate information is traditionally kept in **c_cflag**.
 6098 Applications should be written to presume that this might be the case (and thus not blindly copy
 6099 **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag**
 6100 field absolutely after setting a baud rate is a non-portable action because of this. In general, the

6101 unused parts of the flag fields might be used by the implementation and should not be blindly
6102 copied from the descriptions of one terminal device to another.

6103 **FUTURE DIRECTIONS**

6104 None.

6105 **SEE ALSO**

6106 *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*, the Base Definitions volume of
6107 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<termios.h>**

6108 **CHANGE HISTORY**

6109 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

6110 **NAME**

6111 cfgetospeed — get output baud rate

6112 **SYNOPSIS**

6113 #include <termios.h>

6114 speed_t cfgetospeed(const struct termios *termios_p);

6115 **DESCRIPTION**6116 The *cfgetospeed()* function shall extract the output baud rate from the **termios** structure to which
6117 the *termios_p* argument points.6118 This function shall return exactly the value in the **termios** data structure, without interpretation.6119 **RETURN VALUE**6120 Upon successful completion, *cfgetospeed()* shall return a value of type **speed_t** representing the
6121 output baud rate.6122 **ERRORS**

6123 No errors are defined.

6124 **EXAMPLES**

6125 None.

6126 **APPLICATION USAGE**

6127 None.

6128 **RATIONALE**6129 Refer to *cfgetispeed()*.6130 **FUTURE DIRECTIONS**

6131 None.

6132 **SEE ALSO**6133 *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*, the Base Definitions volume of
6134 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, <**termios.h**>6135 **CHANGE HISTORY**

6136 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

6137 **NAME**

6138 cfsetispeed — set input baud rate

6139 **SYNOPSIS**

6140 #include <termios.h>

6141 int cfsetispeed(struct termios *termios_p, speed_t speed);

6142 **DESCRIPTION**6143 The *cfsetispeed()* function shall set the input baud rate stored in the structure pointed to by
6144 *termios_p* to *speed*.6145 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
6146 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
6147 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
6148 function is called.6149 **RETURN VALUE**6150 Upon successful completion, *cfsetispeed()* shall return 0; otherwise, -1 shall be returned, and
6151 *errno* may be set to indicate the error.6152 **ERRORS**6153 The *cfsetispeed()* function may fail if:6154 [EINVAL] The *speed* value is not a valid baud rate.6155 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
6156 <**termios.h**>.6157 **EXAMPLES**

6158 None.

6159 **APPLICATION USAGE**

6160 None.

6161 **RATIONALE**6162 Refer to *cfgetispeed()*.6163 **FUTURE DIRECTIONS**

6164 None.

6165 **SEE ALSO**6166 *cfgetispeed()*, *cfgetospeed()*, *cfsetospeed()*, *tcsetattr()*, the Base Definitions volume of
6167 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, <**termios.h**>6168 **CHANGE HISTORY**

6169 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

6170 **Issue 6**6171 The following new requirements on POSIX implementations derive from alignment with the
6172 Single UNIX Specification:

- 6173
- The optional setting of *errno* and the [EINVAL] error conditions are added.

6174 **NAME**

6175 cfsetospeed — set output baud rate

6176 **SYNOPSIS**

6177 #include <termios.h>

6178 int cfsetospeed(struct termios *termios_p, speed_t speed);

6179 **DESCRIPTION**6180 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by
6181 *termios_p* to *speed*.6182 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
6183 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
6184 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
6185 function is called.6186 **RETURN VALUE**6187 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*
6188 may be set to indicate the error.6189 **ERRORS**6190 The *cfsetospeed()* function may fail if:6191 [EINVAL] The *speed* value is not a valid baud rate.6192 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
6193 <**termios.h**>.6194 **EXAMPLES**

6195 None.

6196 **APPLICATION USAGE**

6197 None.

6198 **RATIONALE**6199 Refer to *cfgetispeed()*.6200 **FUTURE DIRECTIONS**

6201 None.

6202 **SEE ALSO**6203 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*, the Base Definitions volume of
6204 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, <**termios.h**>6205 **CHANGE HISTORY**

6206 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

6207 **Issue 6**6208 The following new requirements on POSIX implementations derive from alignment with the
6209 Single UNIX Specification:

- 6210
- The optional setting of *errno* and the [EINVAL] error conditions are added.

6211 **NAME**

6212 chdir — change working directory

6213 **SYNOPSIS**

6214 #include <unistd.h>

6215 int chdir(const char *path);

6216 **DESCRIPTION**

6217 The *chdir()* function shall cause the directory named by the pathname pointed to by the *path*
 6218 argument to become the current working directory; that is, the starting point for path searches
 6219 for pathnames not beginning with *'/'*.

6220 **RETURN VALUE**

6221 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current
 6222 working directory shall remain unchanged, and *errno* shall be set to indicate the error.

6223 **ERRORS**6224 The *chdir()* function shall fail if:

6225 [EACCES] Search permission is denied for any component of the pathname.

6226 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 6227 argument.

6228 [ENAMETOOLONG]

6229 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 6230 component is longer than {NAME_MAX}.

6231 [ENOENT] A component of *path* does not name an existing directory or *path* is an empty
 6232 string.

6233 [ENOTDIR] A component of the pathname is not a directory.

6234 The *chdir()* function may fail if:

6235 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 6236 resolution of the *path* argument.

6237 [ENAMETOOLONG]

6238 As a result of encountering a symbolic link in resolution of the *path* argument,
 6239 the length of the substituted pathname string exceeded {PATH_MAX}.

6240 **EXAMPLES**6241 **Changing the Current Working Directory**

6242 The following example makes the value pointed to by **directory**, **/tmp**, the current working
 6243 directory.

6244 #include <unistd.h>

6245 ...

6246 char *directory = "/tmp";

6247 int ret;

6248 ret = chdir (directory);

6249 APPLICATION USAGE

6250 None.

6251 RATIONALE

6252 The *chdir()* function only affects the working directory of the current process.

6253 FUTURE DIRECTIONS

6254 None.

6255 SEE ALSO

6256 *getcwd()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

6257 CHANGE HISTORY

6258 First released in Issue 1. Derived from Issue 1 of the SVID.

6259 Issue 6

6260 The APPLICATION USAGE section is added.

6261 The following new requirements on POSIX implementations derive from alignment with the
6262 Single UNIX Specification:

- 6263 • The [ELOOP] mandatory error condition is added.
- 6264 • A second [ENAMETOOLONG] is added as an optional error condition.

6265 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6266 • The [ELOOP] optional error condition is added.

6267 **NAME**

6268 chmod — change mode of a file

6269 **SYNOPSIS**

6270 #include <sys/stat.h>

6271 int chmod(const char *path, mode_t mode);

6272 **DESCRIPTION**

6273 xSI The *chmod()* function shall change S_ISUID, S_ISGID, S_ISVTX, and the file permission bits of
 6274 the file named by the pathname pointed to by the *path* argument to the corresponding bits in the
 6275 *mode* argument. The application shall ensure that the effective user ID of the process matches the
 6276 owner of the file or the process has appropriate privileges in order to do this.

6277 xSI S_ISUID, S_ISGID, S_ISVTX, and the file permission bits are described in <sys/stat.h>.

6278 If the calling process does not have appropriate privileges, and if the group ID of the file does
 6279 not match the effective group ID or one of the supplementary group IDs and if the file is a
 6280 regular file, bit S_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon
 6281 successful return from *chmod()*.

6282 Additional implementation-defined restrictions may cause the S_ISUID and S_ISGID bits in
 6283 *mode* to be ignored.

6284 The effect on file descriptors for files open at the time of a call to *chmod()* is implementation-
 6285 defined.

6286 Upon successful completion, *chmod()* shall mark for update the *st_ctime* field of the file.

6287 **RETURN VALUE**

6288 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 6289 indicate the error. If -1 is returned, no change to the file mode occurs.

6290 **ERRORS**

6291 The *chmod()* function shall fail if:

6292 [EACCES] Search permission is denied on a component of the path prefix.

6293 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 6294 argument.

6295 [ENAMETOOLONG]

6296 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 6297 component is longer than {NAME_MAX}.

6298 [ENOTDIR] A component of the path prefix is not a directory.

6299 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

6300 [EPERM] The effective user ID does not match the owner of the file and the process
 6301 does not have appropriate privileges.

6302 [EROFS] The named file resides on a read-only file system.

6303 The *chmod()* function may fail if:

6304 [EINTR] A signal was caught during execution of the function.

6305 [EINVAL] The value of the *mode* argument is invalid.

6306 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 6307 resolution of the *path* argument.

6308 [ENAMETOOLONG]
 6309 As a result of encountering a symbolic link in resolution of the *path* argument,
 6310 the length of the substituted pathname strings exceeded {PATH_MAX}.

6311 EXAMPLES

6312 Setting Read Permissions for User, Group, and Others

6313 The following example sets read permissions for the owner, group, and others.

```
6314 #include <sys/stat.h>
6315 const char *path;
6316 ...
6317 chmod(path, S_IRUSR|S_IRGRP|S_IROTH);
```

6318 Setting Read, Write, and Execute Permissions for the Owner Only

6319 The following example sets read, write, and execute permissions for the owner, and no
 6320 permissions for group and others.

```
6321 #include <sys/stat.h>
6322 const char *path;
6323 ...
6324 chmod(path, S_IRWXU);
```

6325 Setting Different Permissions for Owner, Group, and Other

6326 The following example sets owner permissions for CHANGEFILE to read, write, and execute,
 6327 group permissions to read and execute, and other permissions to read.

```
6328 #include <sys/stat.h>
6329 #define CHANGEFILE "/etc/myfile"
6330 ...
6331 chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

6332 Setting and Checking File Permissions

6333 The following example sets the file permission bits for a file named **/home/cnd/mod1**, then calls
 6334 the *stat()* function to verify the permissions.

```
6335 #include <sys/types.h>
6336 #include <sys/stat.h>
6337 int status;
6338 struct stat buffer
6339 ...
6340 chmod("home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
6341 status = stat("home/cnd/mod1", &buffer);
```

6342 APPLICATION USAGE

6343 In order to ensure that the S_ISUID and S_ISGID bits are set, an application requiring this should
 6344 use *stat()* after a successful *chmod()* to verify this.

6345 Any file descriptors currently open by any process on the file could possibly become invalid if
 6346 the mode of the file is changed to a value which would deny access to that process. One

6347 situation where this could occur is on a stateless file system. This behavior will not occur in a
6348 conforming environment.

6349 **RATIONALE**

6350 This volume of IEEE Std 1003.1-2001 specifies that the S_ISGID bit is cleared by *chmod()* on a
6351 regular file under certain conditions. This is specified on the assumption that regular files may
6352 be executed, and the system should prevent users from making executable *setgid()* files perform
6353 with privileges that the caller does not have. On implementations that support execution of
6354 other file types, the S_ISGID bit should be cleared for those file types under the same
6355 circumstances.

6356 Implementations that use the S_ISUID bit to indicate some other function (for example,
6357 mandatory record locking) on non-executable files need not clear this bit on writing. They
6358 should clear the bit for executable files and any other cases where the bit grants special powers
6359 to processes that change the file contents. Similar comments apply to the S_ISGID bit.

6360 **FUTURE DIRECTIONS**

6361 None.

6362 **SEE ALSO**

6363 *chown()*, *mkdir()*, *mkfifo()*, *open()*, *stat()*, *statvfs()*, the Base Definitions volume of
6364 IEEE Std 1003.1-2001, **<sys/stat.h>**, **<sys/types.h>**

6365 **CHANGE HISTORY**

6366 First released in Issue 1. Derived from Issue 1 of the SVID.

6367 **Issue 6**

6368 The following new requirements on POSIX implementations derive from alignment with the
6369 Single UNIX Specification:

6370 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
6371 required for conforming implementations of previous POSIX specifications, it was not
6372 required for UNIX applications.

6373 • The [EINVAL] and [EINTR] optional error conditions are added.

6374 • A second [ENAMETOOLONG] is added as an optional error condition.

6375 The following changes were made to align with the IEEE P1003.1a draft standard:

6376 • The [ELOOP] optional error condition is added.

6377 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

6378 **NAME**

6379 chown — change owner and group of a file

6380 **SYNOPSIS**

6381 #include <unistd.h>

6382 int chown(const char *path, uid_t owner, gid_t group);

6383 **DESCRIPTION**6384 The *chown()* function shall change the user and group ownership of a file.6385 The *path* argument points to a pathname naming a file. The user ID and group ID of the named
6386 file shall be set to the numeric values contained in *owner* and *group*, respectively.6387 Only processes with an effective user ID equal to the user ID of the file or with appropriate
6388 privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for
6389 *path*:

- 6390
- Changing the user ID is restricted to processes with appropriate privileges.
 - Changing the group ID is permitted to a process with an effective user ID equal to the user
6391 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's user
6392 ID or (**uid_t**)−1 and *group* is equal either to the calling process' effective group ID or to one of
6393 its supplementary group IDs.
- 6394

6395 If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of
6396 the file mode are set, and the process does not have appropriate privileges, the set-user-ID
6397 (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful
6398 return from *chown()*. If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`,
6399 or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is
6400 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown()*
6401 function is successfully invoked on a file that is not a regular file and one or more of the
6402 `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID
6403 bits may be cleared.6404 If *owner* or *group* is specified as (**uid_t**)−1 or (**gid_t**)−1, respectively, the corresponding ID of the
6405 file shall not be changed. If both *owner* and *group* are −1, the times need not be updated.6406 Upon successful completion, *chown()* shall mark for update the *st_ctime* field of the file.6407 **RETURN VALUE**6408 Upon successful completion, 0 shall be returned; otherwise, −1 shall be returned and *errno* set to
6409 indicate the error. If −1 is returned, no changes are made in the user ID and group ID of the file.6410 **ERRORS**6411 The *chown()* function shall fail if:

- 6412 [EACCES] Search permission is denied on a component of the path prefix.
-
- 6413 [ELOOP] A loop exists in symbolic links encountered during resolution of the
- path*
-
- 6414 argument.
-
- 6415 [ENAMETOOLONG] The length of the
- path*
- argument exceeds {PATH_MAX} or a pathname
-
- 6416 component is longer than {NAME_MAX}.
-
- 6417 [ENOTDIR] A component of the path prefix is not a directory.
-
- 6418 [ENOENT] A component of
- path*
- does not name an existing file or
- path*
- is an empty string.
-
- 6419

6420 [EPERM] The effective user ID does not match the owner of the file, or the calling
 6421 process does not have appropriate privileges and
 6422 _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

6423 [EROFS] The named file resides on a read-only file system.

6424 The *chown()* function may fail if:

6425 [EIO] An I/O error occurred while reading or writing to the file system.

6426 [EINTR] The *chown()* function was interrupted by a signal which was caught.

6427 [EINVAL] The owner or group ID supplied is not a value supported by the
 6428 implementation.

6429 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 6430 resolution of the *path* argument.

6431 [ENAMETOOLONG]
 6432 As a result of encountering a symbolic link in resolution of the *path* argument,
 6433 the length of the substituted pathname string exceeded {PATH_MAX}.

6434 EXAMPLES

6435 None.

6436 APPLICATION USAGE

6437 Although *chown()* can be used on some implementations by the file owner to change the owner
 6438 and group to any desired values, the only portable use of this function is to change the group of
 6439 a file to the effective GID of the calling process or to a member of its group set.

6440 RATIONALE

6441 System III and System V allow a user to give away files; that is, the owner of a file may change
 6442 its user ID to anything. This is a serious problem for implementations that are intended to meet
 6443 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the
 6444 user ID of a file. Some government agencies (usually not ones concerned directly with security)
 6445 find this limitation too confining. This volume of IEEE Std 1003.1-2001 uses *may* to permit secure
 6446 implementations while not disallowing System V.

6447 System III and System V allow the owner of a file to change the group ID to anything. Version 7
 6448 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to
 6449 change the group ID of a file to its effective group ID or to any of the groups in the list of
 6450 supplementary group IDs, but to no others.

6451 The POSIX.1-1990 standard requires that the *chown()* function invoked by a non-appropriate
 6452 privileged process clear the S_ISGID and the S_ISUID bits for regular files, and permits them to
 6453 be cleared for other types of files. This is so that changes in accessibility do not accidentally
 6454 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-
 6455 executable data files also clears the mandatory file locking bit (shared with S_ISGID), which is
 6456 an extension on many implementations (it first appeared in System V). These bits should only be
 6457 required to be cleared on regular files that have one or more of their execute bits set.

6458 FUTURE DIRECTIONS

6459 None.

6460 SEE ALSO

6461 *chmod()*, *pathconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/types.h**>,
 6462 <**unistd.h**>

6463 **CHANGE HISTORY**

6464 First released in Issue 1. Derived from Issue 1 of the SVID.

6465 **Issue 6**

6466 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 6467 • The wording describing the optional dependency on `_POSIX_CHOWN_RESTRICTED` is
- 6468 restored.
- 6469 • The `[EPERM]` error is restored as an error dependent on `_POSIX_CHOWN_RESTRICTED`.
- 6470 This is since its operand is a pathname and applications should be aware that the error may
- 6471 not occur for that pathname if the file system does not support
- 6472 `_POSIX_CHOWN_RESTRICTED`.

6473 The following new requirements on POSIX implementations derive from alignment with the

6474 Single UNIX Specification:

- 6475 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
- 6476 required for conforming implementations of previous POSIX specifications, it was not
- 6477 required for UNIX applications.
- 6478 • The value for *owner* of `(uid_t)-1` allows the use of `-1` by the owner of a file to change the
- 6479 group ID only. A corresponding change is made for group.
- 6480 • The `[ELOOP]` mandatory error condition is added.
- 6481 • The `[EIO]` and `[EINTR]` optional error conditions are added.
- 6482 • A second `[ENAMETOOLONG]` is added as an optional error condition.

6483 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6484 • Clarification is added that the `S_ISUID` and `S_ISGID` bits do not need to be cleared when the
- 6485 process has appropriate privileges.
- 6486 • The `[ELOOP]` optional error condition is added.

6487 **NAME**

6488 cimag, cimagf, cimagl — complex imaginary functions

6489 **SYNOPSIS**

6490 #include <complex.h>

6491 double cimag(double complex z);

6492 float cimagf(float complex z);

6493 long double cimagl(long double complex z);

6494 **DESCRIPTION**

6495 CX The functionality described on this reference page is aligned with the ISO C standard. Any
6496 conflict between the requirements described here and the ISO C standard is unintentional. This
6497 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6498 These functions shall compute the imaginary part of *z*.6499 **RETURN VALUE**

6500 These functions shall return the imaginary part value (as a real).

6501 **ERRORS**

6502 No errors are defined.

6503 **EXAMPLES**

6504 None.

6505 **APPLICATION USAGE**6506 For a variable *z* of complex type:6507 *z* == creal(*z*) + cimag(*z*)*I6508 **RATIONALE**

6509 None.

6510 **FUTURE DIRECTIONS**

6511 None.

6512 **SEE ALSO**6513 *carg()*, *conj()*, *cproj()*, *creal()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>6514 **CHANGE HISTORY**

6515 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6516 **NAME**

6517 clearerr — clear indicators on a stream

6518 **SYNOPSIS**

6519 #include <stdio.h>

6520 void clearerr(FILE **stream*);6521 **DESCRIPTION**

6522 cx The functionality described on this reference page is aligned with the ISO C standard. Any
6523 conflict between the requirements described here and the ISO C standard is unintentional. This
6524 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6525 The *clearerr()* function shall clear the end-of-file and error indicators for the stream to which
6526 *stream* points.

6527 **RETURN VALUE**6528 The *clearerr()* function shall not return a value.6529 **ERRORS**

6530 No errors are defined.

6531 **EXAMPLES**

6532 None.

6533 **APPLICATION USAGE**

6534 None.

6535 **RATIONALE**

6536 None.

6537 **FUTURE DIRECTIONS**

6538 None.

6539 **SEE ALSO**6540 The Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>6541 **CHANGE HISTORY**

6542 First released in Issue 1. Derived from Issue 1 of the SVID.

6543 **NAME**

6544 clock — report CPU time used

6545 **SYNOPSIS**

6546 #include <time.h>

6547 clock_t clock(void);

6548 **DESCRIPTION**

6549 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 6550 conflict between the requirements described here and the ISO C standard is unintentional. This
 6551 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6552 The *clock()* function shall return the implementation's best approximation to the processor time
 6553 used by the process since the beginning of an implementation-defined era related only to the
 6554 process invocation.

6555 **RETURN VALUE**

6556 To determine the time in seconds, the value returned by *clock()* should be divided by the value
 6557 XSI of the macro CLOCKS_PER_SEC. CLOCKS_PER_SEC is defined to be one million in <time.h>.
 6558 If the processor time used is not available or its value cannot be represented, the function shall
 6559 return the value (clock_t)-1.

6560 **ERRORS**

6561 No errors are defined.

6562 **EXAMPLES**

6563 None.

6564 **APPLICATION USAGE**

6565 In order to measure the time spent in a program, *clock()* should be called at the start of the
 6566 program and its return value subtracted from the value returned by subsequent calls. The value
 6567 returned by *clock()* is defined for compatibility across systems that have clocks with different
 6568 resolutions. The resolution on any particular system need not be to microsecond accuracy.

6569 The value returned by *clock()* may wrap around on some implementations. For example, on a
 6570 machine with 32-bit values for **clock_t**, it wraps after 2 147 seconds or 36 minutes.

6571 **RATIONALE**

6572 None.

6573 **FUTURE DIRECTIONS**

6574 None.

6575 **SEE ALSO**

6576 *asctime()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
 6577 the Base Definitions volume of IEEE Std 1003.1-2001, <time.h>

6578 **CHANGE HISTORY**

6579 First released in Issue 1. Derived from Issue 1 of the SVID.

6580 **NAME**6581 clock_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)6582 **SYNOPSIS**6583 CPT `#include <time.h>`6584 `int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);`

6585

6586 **DESCRIPTION**

6587 The `clock_getcpuclockid()` function shall return the clock ID of the CPU-time clock of the process
6588 specified by *pid*. If the process described by *pid* exists and the calling process has permission,
6589 the clock ID of this clock shall be returned in *clock_id*.

6590 If *pid* is zero, the `clock_getcpuclockid()` function shall return the clock ID of the CPU-time clock of
6591 the process making the call, in *clock_id*.

6592 The conditions under which one process has permission to obtain the CPU-time clock ID of
6593 other processes are implementation-defined.

6594 **RETURN VALUE**

6595 Upon successful completion, `clock_getcpuclockid()` shall return zero; otherwise, an error number
6596 shall be returned to indicate the error.

6597 **ERRORS**

6598 The `clock_getcpuclockid()` function shall fail if:

6599 [EPERM] The requesting process does not have permission to access the CPU-time
6600 clock for the process.

6601 The `clock_getcpuclockid()` function may fail if:

6602 [ESRCH] No process can be found corresponding to the process specified by *pid*.

6603 **EXAMPLES**

6604 None.

6605 **APPLICATION USAGE**

6606 The `clock_getcpuclockid()` function is part of the Process CPU-Time Clocks option and need not
6607 be provided on all implementations.

6608 **RATIONALE**

6609 None.

6610 **FUTURE DIRECTIONS**

6611 None.

6612 **SEE ALSO**

6613 `clock_getres()`, `timer_create()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

6614 **CHANGE HISTORY**

6615 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

6616 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

6617 **NAME**6618 clock_getres, clock_gettime, clock_settime — clock and timer functions (**REALTIME**)6619 **SYNOPSIS**

6620 TMR #include <time.h>

```

6621 int clock_getres(clockid_t clock_id, struct timespec *res);
6622 int clock_gettime(clockid_t clock_id, struct timespec *tp);
6623 int clock_settime(clockid_t clock_id, const struct timespec *tp);
6624

```

6625 **DESCRIPTION**

6626 The *clock_getres()* function shall return the resolution of any clock. Clock resolutions are
 6627 implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the
 6628 resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL,
 6629 the clock resolution is not returned. If the *time* argument of *clock_settime()* is not a multiple of *res*,
 6630 then the value is truncated to a multiple of *res*.

6631 The *clock_gettime()* function shall return the current value *tp* for the specified clock, *clock_id*.

6632 The *clock_settime()* function shall set the specified clock, *clock_id*, to the value specified by *tp*.
 6633 Time values that are between two consecutive non-negative integer multiples of the resolution
 6634 of the specified clock shall be truncated down to the smaller multiple of the resolution.

6635 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that
 6636 is meaningful only within a process). All implementations shall support a *clock_id* of
 6637 CLOCK_REALTIME as defined in <time.h>. This clock represents the realtime clock for the
 6638 system. For this clock, the values returned by *clock_gettime()* and specified by *clock_settime()*
 6639 represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation
 6640 may also support additional clocks. The interpretation of time values for these clocks is
 6641 unspecified.

6642 If the value of the CLOCK_REALTIME clock is set via *clock_settime()*, the new value of the clock
 6643 shall be used to determine the time of expiration for absolute time services based upon the
 6644 CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the
 6645 absolute time requested at the invocation of such a time service is before the new value of the
 6646 clock, the time service shall expire immediately as if the clock had reached the requested time
 6647 normally.

6648 Setting the value of the CLOCK_REALTIME clock via *clock_settime()* shall have no effect on
 6649 threads that are blocked waiting for a relative time service based upon this clock, including the
 6650 *nanosleep()* function; nor on the expiration of relative timers based upon this clock.
 6651 Consequently, these time services shall expire when the requested relative interval elapses,
 6652 independently of the new or old value of the clock.

6653 MON If the Monotonic Clock option is supported, all implementations shall support a *clock_id* of
 6654 CLOCK_MONOTONIC defined in <time.h>. This clock represents the monotonic clock for the
 6655 system. For this clock, the value returned by *clock_gettime()* represents the amount of time (in
 6656 seconds and nanoseconds) since an unspecified point in the past (for example, system start-up
 6657 time, or the Epoch). This point does not change after system start-up time. The value of the
 6658 CLOCK_MONOTONIC clock cannot be set via *clock_settime()*. This function shall fail if it is
 6659 invoked with a *clock_id* argument of CLOCK_MONOTONIC.

6660 The effect of setting a clock via *clock_settime()* on armed per-process timers associated with a
 6661 clock other than CLOCK_REALTIME is implementation-defined.

6662 CS If the value of the CLOCK_REALTIME clock is set via *clock_settime()*, the new value of the clock
 6663 shall be used to determine the time at which the system shall awaken a thread blocked on an

absolute *clock_nanosleep()* call based upon the `CLOCK_REALTIME` clock. If the absolute time requested at the invocation of such a time service is before the new value of the clock, the call shall return immediately as if the clock had reached the requested time normally.

Setting the value of the `CLOCK_REALTIME` clock via *clock_settime()* shall have no effect on any thread that is blocked on a relative *clock_nanosleep()* call. Consequently, the call shall return when the requested relative interval elapses, independently of the new or old value of the clock.

The appropriate privilege to set a particular clock is implementation-defined.

CPT If `_POSIX_CPUTIME` is defined, implementations shall support clock ID values obtained by invoking *clock_getcpuclockid()*, which represent the CPU-time clock of a given process. Implementations shall also support the special **clockid_t** value `CLOCK_PROCESS_CPUTIME_ID`, which represents the CPU-time clock of the calling process when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values returned by *clock_gettime()* and specified by *clock_settime()* represent the amount of execution time of the process associated with the clock. Changing the value of a CPU-time clock via *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see **Scheduling Policies** (on page 44)).

TCT If `_POSIX_THREAD_CPUTIME` is defined, implementations shall support clock ID values obtained by invoking *pthread_getcpuclockid()*, which represent the CPU-time clock of a given thread. Implementations shall also support the special **clockid_t** value `CLOCK_THREAD_CPUTIME_ID`, which represents the CPU-time clock of the calling thread when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values returned by *clock_gettime()* and specified by *clock_settime()* shall represent the amount of execution time of the thread associated with the clock. Changing the value of a CPU-time clock via *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see **Scheduling Policies** (on page 44)).

6689 RETURN VALUE

A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that an error occurred, and *errno* shall be set to indicate the error.

6692 ERRORS

The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions shall fail if:

[EINVAL] The *clock_id* argument does not specify a known clock.

The *clock_settime()* function shall fail if:

[EINVAL] The *tp* argument to *clock_settime()* is outside the range for the given clock ID.

[EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.

MON [EINVAL] The value of the *clock_id* argument is `CLOCK_MONOTONIC`.

The *clock_settime()* function may fail if:

[EPERM] The requesting process does not have the appropriate privilege to set the specified clock.

6703 **EXAMPLES**

6704 None.

6705 **APPLICATION USAGE**

6706 These functions are part of the Timers option and need not be available on all implementations.

6707 Note that the absolute value of the monotonic clock is meaningless (because its origin is
 6708 arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the
 6709 fact that the value of this clock is never set and, therefore, that time intervals measured with this
 6710 clock will not be affected by calls to *clock_settime()*.

6711 **RATIONALE**

6712 None.

6713 **FUTURE DIRECTIONS**

6714 None.

6715 **SEE ALSO**

6716 *clock_getcpuclockid()*, *clock_nanosleep()*, *ctime()*, *mq_timedreceive()*, *mq_timedsend()*, *nanosleep()*,
 6717 *pthread_mutex_timedlock()*, *sem_timedwait()*, *time()*, *timer_create()*, *timer_getoverrun()*, the Base
 6718 Definitions volume of IEEE Std 1003.1-2001, <time.h>

6719 **CHANGE HISTORY**

6720 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

6721 **Issue 6**

6722 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 6723 implementation does not support the Timers option.

6724 The APPLICATION USAGE section is added.

6725 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6726 • Clarification is added of the effect of resetting the clock resolution.

6727 CPU-time clocks and the *clock_getcpuclockid()* function are added for alignment with
 6728 IEEE Std 1003.1d-1999.

6729 The following changes are added for alignment with IEEE Std 1003.1j-2000:

- 6730 • The DESCRIPTION is updated as follows:

- 6731 — The value returned by *clock_gettime()* for CLOCK_MONOTONIC is specified.

- 6732 — The *clock_settime()* function failing for CLOCK_MONOTONIC is specified.

- 6733 — The effects of *clock_settime()* on the *clock_nanosleep()* function with respect to
 6734 CLOCK_REALTIME are specified.

- 6735 • An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime()* fails for
 6736 CLOCK_MONOTONIC.

- 6737 • The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not
 6738 and shall not be set by *clock_settime()* since the absolute value of the CLOCK_MONOTONIC
 6739 clock is meaningless.

- 6740 • The *clock_nanosleep()*, *mq_timedreceive()*, *mq_timedsend()*, *pthread_mutex_timedlock()*,
 6741 *sem_timedwait()*, *timer_create()*, and *timer_settime()* functions are added to the SEE ALSO
 6742 section.

6743 **NAME**6744 clock_nanosleep — high resolution sleep with specifiable clock (**ADVANCED REALTIME**)6745 **SYNOPSIS**6746 `cs #include <time.h>`

```
6747 int clock_nanosleep(clockid_t clock_id, int flags,
6748     const struct timespec *rqtp, struct timespec *rmtp);
6749
```

6750 **DESCRIPTION**

6751 If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the *clock_nanosleep()* function shall
 6752 cause the current thread to be suspended from execution until either the time interval specified
 6753 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to
 6754 invoke a signal-catching function, or the process is terminated. The clock used to measure the
 6755 time shall be the clock specified by *clock_id*.

6756 If the flag `TIMER_ABSTIME` is set in the *flags* argument, the *clock_nanosleep()* function shall
 6757 cause the current thread to be suspended from execution until either the time value of the clock
 6758 specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is
 6759 delivered to the calling thread and its action is to invoke a signal-catching function, or the
 6760 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or
 6761 equal to the time value of the specified clock, then *clock_nanosleep()* shall return immediately
 6762 and the calling process shall not be suspended.

6763 The suspension time caused by this function may be longer than requested because the
 6764 argument value is rounded up to an integer multiple of the sleep resolution, or because of the
 6765 scheduling of other activity by the system. But, except for the case of being interrupted by a
 6766 signal, the suspension time for the relative *clock_nanosleep()* function (that is, with the
 6767 `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as
 6768 measured by the corresponding clock. The suspension for the absolute *clock_nanosleep()* function
 6769 (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the
 6770 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being
 6771 interrupted by a signal.

6772 The use of the *clock_nanosleep()* function shall have no effect on the action or blockage of any
 6773 signal.

6774 The *clock_nanosleep()* function shall fail if the *clock_id* argument refers to the CPU-time clock of
 6775 the calling thread. It is unspecified whether *clock_id* values of other CPU-time clocks are
 6776 allowed.

6777 **RETURN VALUE**

6778 If the *clock_nanosleep()* function returns because the requested time has elapsed, its return value
 6779 shall be zero.

6780 If the *clock_nanosleep()* function returns because it has been interrupted by a signal, it shall return
 6781 the corresponding error value. For the relative *clock_nanosleep()* function, if the *rmtp* argument is
 6782 non-NULL, the **timespec** structure referenced by it shall be updated to contain the amount of
 6783 time remaining in the interval (the requested time minus the time actually slept). If the *rmtp*
 6784 argument is NULL, the remaining time is not returned. The absolute *clock_nanosleep()* function
 6785 has no effect on the structure referenced by *rmtp*.

6786 If *clock_nanosleep()* fails, it shall return the corresponding error value.

6787 ERRORS

6788 The *clock_nanosleep()* function shall fail if:

6789	[EINTR]	The <i>clock_nanosleep()</i> function was interrupted by a signal.
6790	[EINVAL]	The <i>rtp</i> argument specified a nanosecond value less than zero or greater than or equal to 1 000 million; or the <code>TIMER_ABSTIME</code> flag was specified in <i>flags</i> and the <i>rtp</i> argument is outside the range for the clock specified by <i>clock_id</i> ; or the <i>clock_id</i> argument does not specify a known clock, or specifies the CPU-time clock of the calling thread.
6795	[ENOTSUP]	The <i>clock_id</i> argument specifies a clock for which <i>clock_nanosleep()</i> is not supported, such as a CPU-time clock.

6797 EXAMPLES

6798 None.

6799 APPLICATION USAGE

6800 Calling *clock_nanosleep()* with the value `TIMER_ABSTIME` not set in the *flags* argument and with a *clock_id* of `CLOCK_REALTIME` is equivalent to calling *nanosleep()* with the same *rtp* and *rmtp* arguments.

6803 RATIONALE

6804 The *nanosleep()* function specifies that the system-wide clock `CLOCK_REALTIME` is used to measure the elapsed time for this time service. However, with the introduction of the monotonic clock `CLOCK_MONOTONIC` a new relative sleep function is needed to allow an application to take advantage of the special characteristics of this clock.

6808 There are many applications in which a process needs to be suspended and then activated multiple times in a periodic way; for example, to poll the status of a non-interrupting device or to refresh a display device. For these cases, it is known that precise periodic activation cannot be achieved with a relative *sleep()* or *nanosleep()* function call. Suppose, for example, a periodic process that is activated at time *T*₀, executes for a while, and then wants to suspend itself until time *T*₀+*T*, the period being *T*. If this process wants to use the *nanosleep()* function, it must first call *clock_gettime()* to get the current time, then calculate the difference between the current time and *T*₀+*T* and, finally, call *nanosleep()* using the computed interval. However, the process could be preempted by a different process between the two function calls, and in this case the interval computed would be wrong; the process would wake up later than desired. This problem would not occur with the absolute *clock_nanosleep()* function, since only one function call would be necessary to suspend the process until the desired time. In other cases, however, a relative sleep is needed, and that is why both functionalities are required.

6821 Although it is possible to implement periodic processes using the timers interface, this implementation would require the use of signals, and the reservation of some signal numbers. In this regard, the reasons for including an absolute version of the *clock_nanosleep()* function in IEEE Std 1003.1-2001 are the same as for the inclusion of the relative *nanosleep()*.

6825 It is also possible to implement precise periodic processes using *pthread_cond_timedwait()*, in which an absolute timeout is specified that takes effect if the condition variable involved is never signaled. However, the use of this interface is unnatural, and involves performing other operations on mutexes and condition variables that imply an unnecessary overhead. Furthermore, *pthread_cond_timedwait()* is not available in implementations that do not support threads.

6831 Although the interface of the relative and absolute versions of the new high resolution sleep service is the same *clock_nanosleep()* function, the *rmtp* argument is only used in the relative sleep. This argument is needed in the relative *clock_nanosleep()* function to reissue the function

6834 call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep()* function
6835 call; if the call is interrupted by a signal, the absolute *clock_nanosleep()* function can be invoked
6836 again with the same *rqtp* argument used in the interrupted call.

6837 **FUTURE DIRECTIONS**

6838 None.

6839 **SEE ALSO**

6840 *clock_getres()*, *nanosleep()*, *pthread_cond_timedwait()*, *sleep()*, the Base Definitions volume of
6841 IEEE Std 1003.1-2001, <**time.h**>

6842 **CHANGE HISTORY**

6843 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

6844 **NAME**6845 clock_settime — clock and timer functions (**REALTIME**)6846 **SYNOPSIS**6847 TMR `#include <time.h>`6848 `int clock_settime(clockid_t clock_id, const struct timespec *tp);`

6849

6850 **DESCRIPTION**6851 Refer to *clock_getres()*.

6852 NAME

6853 clog, clogf, clogl — complex natural logarithm functions

6854 SYNOPSIS

6855 #include <complex.h>

6856 double complex clog(double complex z);

6857 float complex clogf(float complex z);

6858 long double complex clogl(long double complex z);

6859 DESCRIPTION

6860 cx The functionality described on this reference page is aligned with the ISO C standard. Any
6861 conflict between the requirements described here and the ISO C standard is unintentional. This
6862 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6863 These functions shall compute the complex natural (base e) logarithm of z , with a branch cut
6864 along the negative real axis.

6865 RETURN VALUE

6866 These functions shall return the complex natural logarithm value, in the range of a strip
6867 mathematically unbounded along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary
6868 axis.

6869 ERRORS

6870 No errors are defined.

6871 EXAMPLES

6872 None.

6873 APPLICATION USAGE

6874 None.

6875 RATIONALE

6876 None.

6877 FUTURE DIRECTIONS

6878 None.

6879 SEE ALSO

6880 *cexp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>

6881 CHANGE HISTORY

6882 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6883 **NAME**

6884 close — close a file descriptor

6885 **SYNOPSIS**

6886 #include <unistd.h>

6887 int close(int *fildes*);6888 **DESCRIPTION**

6889 The *close()* function shall deallocate the file descriptor indicated by *fildes*. To deallocate means
 6890 to make the file descriptor available for return by subsequent calls to *open()* or other functions
 6891 that allocate file descriptors. All outstanding record locks owned by the process on the file
 6892 associated with the file descriptor shall be removed (that is, unlocked).

6893 If *close()* is interrupted by a signal that is to be caught, it shall return -1 with *errno* set to [EINTR]
 6894 and the state of *fildes* is unspecified. If an I/O error occurred while reading from or writing to the
 6895 file system during *close()*, it may return -1 with *errno* set to [EIO]; if this error is returned, the
 6896 state of *fildes* is unspecified.

6897 When all file descriptors associated with a pipe or FIFO special file are closed, any data
 6898 remaining in the pipe or FIFO shall be discarded.

6899 When all file descriptors associated with an open file description have been closed, the open file
 6900 description shall be freed.

6901 If the link count of the file is 0, when all file descriptors associated with the file are closed, the
 6902 space occupied by the file shall be freed and the file shall no longer be accessible.

6903 XSR If a STREAMS-based *fildes* is closed and the calling process was previously registered to receive
 6904 a SIGPOLL signal for events associated with that STREAM, the calling process shall be
 6905 unregistered for events associated with the STREAM. The last *close()* for a STREAM shall cause
 6906 the STREAM associated with *fildes* to be dismantled. If O_NONBLOCK is not set and there have
 6907 been no signals posted for the STREAM, and if there is data on the module's write queue, *close()*
 6908 shall wait for an unspecified time (for each module and driver) for any output to drain before
 6909 dismantling the STREAM. The time delay can be changed via an I_SETCLTIME *ioctl()* request. If
 6910 the O_NONBLOCK flag is set, or if there are any pending signals, *close()* shall not wait for
 6911 output to drain, and shall dismantle the STREAM immediately.

6912 If the implementation supports STREAMS-based pipes, and *fildes* is associated with one end of a
 6913 pipe, the last *close()* shall cause a hangup to occur on the other end of the pipe. In addition, if the
 6914 other end of the pipe has been named by *fattach()*, then the last *close()* shall force the named end
 6915 to be detached by *fdetach()*. If the named end has no open file descriptors associated with it and
 6916 gets detached, the STREAM associated with that end shall also be dismantled.

6917 XSI If *fildes* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal
 6918 shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is
 6919 the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal
 6920 flushes all queued input and output.

6921 XSR If *fildes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message
 6922 may be sent to the master.

6923 AIO When there is an outstanding cancelable asynchronous I/O operation against *fildes* when *close()*
 6924 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes
 6925 as if the *close()* operation had not yet occurred. All operations that are not canceled shall
 6926 complete as if the *close()* blocked until the operations completed. The *close()* operation itself
 6927 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and
 6928 which I/O operation may be canceled upon *close()*, is implementation-defined.

6929 MF|SHM If a shared memory object or a memory mapped file remains referenced at the last close (that is,
 6930 a process has it mapped), then the entire contents of the memory object shall persist until the
 6931 memory object becomes unreferenced. If this is the last close of a shared memory object or a
 6932 memory mapped file and the close results in the memory object becoming unreferenced, and the
 6933 memory object has been unlinked, then the memory object shall be removed.

6934 If *fildest* refers to a socket, *close()* shall cause the socket to be destroyed. If the socket is in
 6935 connection-mode, and the `SO_LINGER` option is set for the socket with non-zero linger time,
 6936 and the socket has untransmitted data, then *close()* shall block for up to the current linger
 6937 interval until all data is transmitted.

6938 RETURN VALUE

6939 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 6940 indicate the error.

6941 ERRORS

6942 The *close()* function shall fail if:

6943 [EBADF] The *fildest* argument is not a valid file descriptor.

6944 [EINTR] The *close()* function was interrupted by a signal.

6945 The *close()* function may fail if:

6946 [EIO] An I/O error occurred while reading from or writing to the file system.

6947 EXAMPLES

6948 Reassigning a File Descriptor

6949 The following example closes the file descriptor associated with standard output for the current
 6950 process, re-assigns standard output to a new file descriptor, and closes the original file
 6951 descriptor to clean up. This example assumes that the file descriptor 0 (which is the descriptor
 6952 for standard input) is not closed.

```
6953 #include <unistd.h>
6954 ...
6955 int pfd;
6956 ...
6957 close(1);
6958 dup(pfd);
6959 close(pfd);
6960 ...
```

6961 Incidentally, this is exactly what could be achieved using:

```
6962 dup2(pfd, 1);
6963 close(pfd);
```

6964 Closing a File Descriptor

6965 In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is
 6966 made to associate that file descriptor with a stream.

```
6967 #include <stdio.h>
6968 #include <unistd.h>
6969 #include <stdlib.h>
```

```

6970     #define LOCKFILE "/etc/ptmp"
6971     ...
6972     int pfd;
6973     FILE *fpfd;
6974     ...
6975     if ((fpfd = fdopen (pfd, "w")) == NULL) {
6976         close(pfd);
6977         unlink(LOCKFILE);
6978         exit(1);
6979     }
6980     ...

```

6981 APPLICATION USAGE

6982 An application that had used the *stdio* routine *fopen()* to open a file should use the
 6983 corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no
 6984 longer exists, since the integer corresponding to it no longer refers to a file.

6985 RATIONALE

6986 The use of interruptible device close routines should be discouraged to avoid problems with the
 6987 implicit closes of file descriptors by *exec* and *exit()*. This volume of IEEE Std 1003.1-2001 only
 6988 intends to permit such behavior by specifying the [EINTR] error condition.

6989 FUTURE DIRECTIONS

6990 None.

6991 SEE ALSO

6992 Section 2.6 (on page 38), *fattach()*, *fclose()*, *fdetach()*, *fopen()*, *ioctl()*, *open()*, the Base Definitions
 6993 volume of IEEE Std 1003.1-2001, <**unistd.h**>

6994 CHANGE HISTORY

6995 First released in Issue 1. Derived from Issue 1 of the SVID.

6996 Issue 5

6997 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

6998 Issue 6

6999 The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the
 7000 XSI STREAMS Option Group.

7001 The following new requirements on POSIX implementations derive from alignment with the
 7002 Single UNIX Specification:

- 7003 • The [EIO] error condition is added as an optional error.
- 7004 • The DESCRIPTION is updated to describe the state of the *fildev* file descriptor as unspecified
 7005 if an I/O error occurs and an [EIO] error condition is returned.

7006 Text referring to sockets is added to the DESCRIPTION.

7007 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 7008 shared memory objects and memory mapped files (and not typed memory objects) are the types
 7009 of memory objects to which the paragraph on last closes applies.

7010 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 12 is applied, correcting the XSH shaded
 7011 text relating to the master side of a pseudo-terminal. The reason for the change is that the
 7012 behavior of pseudo-terminals and regular terminals should be as much alike as possible in this
 7013 case; the change achieves that and matches historical behavior.

7014 **NAME**

7015 closedir — close a directory stream

7016 **SYNOPSIS**

7017 #include <dirent.h>

7018 int closedir(DIR *dirp);

7019 **DESCRIPTION**

7020 The *closedir()* function shall close the directory stream referred to by the argument *dirp*. Upon
 7021 return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file
 7022 descriptor is used to implement type **DIR**, that file descriptor shall be closed.

7023 **RETURN VALUE**

7024 Upon successful completion, *closedir()* shall return 0; otherwise, -1 shall be returned and *errno*
 7025 set to indicate the error.

7026 **ERRORS**7027 The *closedir()* function may fail if:7028 [EBADF] The *dirp* argument does not refer to an open directory stream.7029 [EINTR] The *closedir()* function was interrupted by a signal.7030 **EXAMPLES**7031 **Closing a Directory Stream**7032 The following program fragment demonstrates how the *closedir()* function is used.

```

7033       ...
7034       DIR *dir;
7035       struct dirent *dp;
7036       ...
7037       if ((dir = opendir(".")) == NULL) {
7038       ...
7039       }
7040       while ((dp = readdir(dir)) != NULL) {
7041       ...
7042       }
7043       closedir(dir);
7044       ...
```

7045 **APPLICATION USAGE**

7046 None.

7047 **RATIONALE**

7048 None.

7049 **FUTURE DIRECTIONS**

7050 None.

7051 **SEE ALSO**7052 *opendir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**dirent.h**>

7053 **CHANGE HISTORY**

7054 First released in Issue 2.

7055 **Issue 6**7056 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.7057 The following new requirements on POSIX implementations derive from alignment with the
7058 Single UNIX Specification:

- 7059 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
7060 required for conforming implementations of previous POSIX specifications, it was not
7061 required for UNIX applications.
- 7062 • The [EINTR] error condition is added as an optional error condition.

NAME

closelog, openlog, setlogmask, syslog — control system log

SYNOPSIS

```
#include <syslog.h>

void closelog(void);
void openlog(const char *ident, int logopt, int facility);
int setlogmask(int maskpri);
void syslog(int priority, const char *message, ... /* arguments */);
```

DESCRIPTION

The *syslog()* function shall send a message to an implementation-defined logging facility, which may log it in an implementation-defined system log, write it to the system console, forward it to a list of users, or forward it to the logging facility on another host over the network. The logged message shall include a message header and a message body. The message header contains at least a timestamp and a tag string.

The message body is generated from the *message* and following arguments in the same manner as if these were arguments to *printf()*, except that the additional conversion specification *%m* shall be recognized; it shall convert no arguments, shall cause the output of the error message string associated with the value of *errno* on entry to *syslog()*, and may be mixed with argument specifications of the "*%n\$*" form. If a complete conversion specification with the *m* conversion specifier character is not just *%m*, the behavior is undefined. A trailing *<newline>* may be added if needed.

Values of the *priority* argument are formed by OR'ing together a severity-level value and an optional facility value. If no facility value is specified, the current default facility value is used.

Possible values of severity level include:

LOG_EMERG	A panic condition.
LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
LOG_CRIT	Critical conditions, such as hard device errors.
LOG_ERR	Errors.
LOG_WARNING	Warning messages.
LOG_NOTICE	Conditions that are not error conditions, but that may require special handling.
LOG_INFO	Informational messages.
LOG_DEBUG	Messages that contain information normally of use only when debugging a program.

The facility indicates the application or system component generating the message. Possible facility values include:

LOG_USER	Messages generated by arbitrary processes. This is the default facility identifier if none is specified.
LOG_LOCAL0	Reserved for local use.

7105	LOG_LOCAL1	Reserved for local use.
7106	LOG_LOCAL2	Reserved for local use.
7107	LOG_LOCAL3	Reserved for local use.
7108	LOG_LOCAL4	Reserved for local use.
7109	LOG_LOCAL5	Reserved for local use.
7110	LOG_LOCAL6	Reserved for local use.
7111	LOG_LOCAL7	Reserved for local use.
7112	The <i>openlog()</i> function shall set process attributes that affect subsequent calls to <i>syslog()</i> . The	
7113	<i>ident</i> argument is a string that is prepended to every message. The <i>logopt</i> argument indicates	
7114	logging options. Values for <i>logopt</i> are constructed by a bitwise-inclusive OR of zero or more of	
7115	the following:	
7116	LOG_PID	Log the process ID with each message. This is useful for identifying specific
7117		processes.
7118	LOG_CONS	Write messages to the system console if they cannot be sent to the logging
7119		facility. The <i>syslog()</i> function ensures that the process does not acquire the
7120		console as a controlling terminal in the process of writing the message.
7121	LOG_NDELAY	Open the connection to the logging facility immediately. Normally the open is
7122		delayed until the first message is logged. This is useful for programs that need
7123		to manage the order in which file descriptors are allocated.
7124	LOG_ODELAY	Delay open until <i>syslog()</i> is called.
7125	LOG_NOWAIT	Do not wait for child processes that may have been created during the course
7126		of logging the message. This option should be used by processes that enable
7127		notification of child termination using SIGCHLD, since <i>syslog()</i> may
7128		otherwise block waiting for a child whose exit status has already been
7129		collected.
7130	The <i>facility</i> argument encodes a default facility to be assigned to all messages that do not have	
7131	an explicit facility already encoded. The initial default facility is LOG_USER.	
7132	The <i>openlog()</i> and <i>syslog()</i> functions may allocate a file descriptor. It is not necessary to call	
7133	<i>openlog()</i> prior to calling <i>syslog()</i> .	
7134	The <i>closelog()</i> function shall close any open file descriptors allocated by previous calls to	
7135	<i>openlog()</i> or <i>syslog()</i> .	
7136	The <i>setlogmask()</i> function shall set the log priority mask for the current process to <i>maskpri</i> and	
7137	return the previous mask. If the <i>maskpri</i> argument is 0, the current log mask is not modified.	
7138	Calls by the current process to <i>syslog()</i> with a priority not set in <i>maskpri</i> shall be rejected. The	
7139	default log mask allows all priorities to be logged. A call to <i>openlog()</i> is not required prior to	
7140	calling <i>setlogmask()</i> .	
7141	Symbolic constants for use as values of the <i>logopt</i> , <i>facility</i> , <i>priority</i> , and <i>maskpri</i> arguments are	
7142	defined in the <syslog.h> header.	
7143	RETURN VALUE	
7144	The <i>setlogmask()</i> function shall return the previous log priority mask. The <i>closelog()</i> , <i>openlog()</i> ,	
7145	and <i>syslog()</i> functions shall not return a value.	

7146 **ERRORS**

7147 No errors are defined.

7148 **EXAMPLES**

7149 **Using openlog()**

7150 The following example causes subsequent calls to *syslog()* to log the process ID with each message, and to write messages to the system console if they cannot be sent to the logging facility.

```
7153       #include <syslog.h>

7154       char *ident = "Process demo";
7155       int logopt = LOG_PID | LOG_CONS;
7156       int facility = LOG_USER;
7157       ...
7158       openlog(ident, logopt, facility);
```

7159 **Using setlogmask()**

7160 The following example causes subsequent calls to *syslog()* to accept error messages, and to reject all other messages.

```
7162       #include <syslog.h>

7163       int result;
7164       int mask = LOG_MASK (LOG_ERR);
7165       ...
7166       result = setlogmask(mask);
```

7167 **Using syslog**

7168 The following example sends the message "This is a message" to the default logging facility, marking the message as an error message generated by random processes.

```
7170       #include <syslog.h>

7171       char *message = "This is a message";
7172       int priority = LOG_ERR | LOG_USER;
7173       ...
7174       syslog(priority, message);
```

7175 **APPLICATION USAGE**

7176 None.

7177 **RATIONALE**

7178 None.

7179 **FUTURE DIRECTIONS**

7180 None.

7181 **SEE ALSO**

7182 *printf()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<syslog.h>**

7183 **CHANGE HISTORY**

7184 First released in Issue 4, Version 2.

7185 **Issue 5**

7186 Moved from X/OPEN UNIX extension to BASE.

7187 **Issue 6**7188 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES
7189 section. |

7190 **NAME**

7191 confstr — get configurable variables

7192 **SYNOPSIS**

7193 #include <unistd.h>

7194 size_t confstr(int name, char *buf, size_t len);

7195 **DESCRIPTION**7196 The *confstr()* function shall return configuration-defined string values. Its use and purpose are
7197 similar to *sysconf()*, but it is used where string values rather than numeric values are returned.7198 The *name* argument represents the system variable to be queried. The implementation shall
7199 support the following name values, defined in <unistd.h>. It may support others:

7200 _CS_PATH
 7201 _CS_POSIX_V6_ILP32_OFF32_CFLAGS
 7202 _CS_POSIX_V6_ILP32_OFF32_LDFLAGS
 7203 _CS_POSIX_V6_ILP32_OFF32_LIBS
 7204 _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
 7205 _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
 7206 _CS_POSIX_V6_ILP32_OFFBIG_LIBS
 7207 _CS_POSIX_V6_LP64_OFF64_CFLAGS
 7208 _CS_POSIX_V6_LP64_OFF64_LDFLAGS
 7209 _CS_POSIX_V6_LP64_OFF64_LIBS
 7210 _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
 7211 _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
 7212 _CS_POSIX_V6_LPBIG_OFFBIG_LIBS
 7213 _CS_POSIX_V6_WIDTH_RESTRICTED_ENVS
 7214 XSI _CS_XBS5_ILP32_OFF32_CFLAGS (**LEGACY**)
 7215 _CS_XBS5_ILP32_OFF32_LDFLAGS (**LEGACY**)
 7216 _CS_XBS5_ILP32_OFF32_LIBS (**LEGACY**)
 7217 _CS_XBS5_ILP32_OFF32_LINTFLAGS (**LEGACY**)
 7218 _CS_XBS5_ILP32_OFFBIG_CFLAGS (**LEGACY**)
 7219 _CS_XBS5_ILP32_OFFBIG_LDFLAGS (**LEGACY**)
 7220 _CS_XBS5_ILP32_OFFBIG_LIBS (**LEGACY**)
 7221 _CS_XBS5_ILP32_OFFBIG_LINTFLAGS (**LEGACY**)
 7222 _CS_XBS5_LP64_OFF64_CFLAGS (**LEGACY**)
 7223 _CS_XBS5_LP64_OFF64_LDFLAGS (**LEGACY**)
 7224 _CS_XBS5_LP64_OFF64_LIBS (**LEGACY**)
 7225 _CS_XBS5_LP64_OFF64_LINTFLAGS (**LEGACY**)
 7226 _CS_XBS5_LPBIG_OFFBIG_CFLAGS (**LEGACY**)
 7227 _CS_XBS5_LPBIG_OFFBIG_LDFLAGS (**LEGACY**)
 7228 _CS_XBS5_LPBIG_OFFBIG_LIBS (**LEGACY**)
 7229 _CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (**LEGACY**)

7230

7231 If *len* is not 0, and if *name* has a configuration-defined value, *confstr()* shall copy that value into
 7232 the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes,
 7233 including the terminating null, then *confstr()* shall truncate the string to *len*–1 bytes and null-
 7234 terminate the result. The application can detect that the string was truncated by comparing the
 7235 value returned by *confstr()* with *len*.

7236 If *len* is 0 and *buf* is a null pointer, then *confstr()* shall still return the integer value as defined
 7237 below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is
 7238 unspecified.

7239 If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

7240 `confstr(_CS_PATH, buf, sizeof(buf))`

7241 can be used as a value of the *PATH* environment variable that accesses all of the standard
7242 utilities of IEEE Std 1003.1-2001, if the return value is less than or equal to *sizeof(buf)*.

7243 RETURN VALUE

7244 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be
7245 needed to hold the entire configuration-defined value including the terminating null. If this
7246 return value is greater than *len*, the string returned in *buf* is truncated.

7247 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

7248 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*
7249 unchanged.

7250 ERRORS

7251 The *confstr()* function shall fail if:

7252 [EINVAL] The value of the *name* argument is invalid.

7253 EXAMPLES

7254 None.

7255 APPLICATION USAGE

7256 An application can distinguish between an invalid *name* parameter value and one that
7257 corresponds to a configurable variable that has no configuration-defined value by checking if
7258 *errno* is modified. This mirrors the behavior of *sysconf()*.

7259 The original need for this function was to provide a way of finding the configuration-defined
7260 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to
7261 include directories that could contain utilities replacing the standard utilities in the Shell and
7262 Utilities volume of IEEE Std 1003.1-2001, applications need a way to determine the system-
7263 supplied *PATH* environment variable value that contains the correct search path for the standard
7264 utilities.

7265 An application could use:

7266 `confstr(name, (char *)NULL, (size_t)0)`

7267 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to
7268 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,
7269 static buffer that is big enough to hold most answers (perhaps 512 or 1024 bytes), but then use
7270 *malloc()* to allocate a larger buffer if it finds that this is too small.

7271 RATIONALE

7272 Application developers can normally determine any configuration variable by means of reading
7273 from the stream opened by a call to:

7274 `popen("command -p getconf variable", "r");`

7275 The *confstr()* function with a *name* argument of *_CS_PATH* returns a string that can be used as a
7276 *PATH* environment variable setting that will reference the standard shell and utilities as
7277 described in the Shell and Utilities volume of IEEE Std 1003.1-2001.

7278 The *confstr()* function copies the returned string into a buffer supplied by the application instead
7279 of returning a pointer to a string. This allows a cleaner function in some implementations (such
7280 as those with lightweight threads) and resolves questions about when the application must copy
7281 the string returned.

7282 **FUTURE DIRECTIONS**

7283 None.

7284 **SEE ALSO**

7285 `pathconf()`, `sysconf()`, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>, the Shell
7286 and Utilities volume of IEEE Std 1003.1-2001, *c99*

7287 **CHANGE HISTORY**

7288 First released in Issue 4. Derived from the ISO POSIX-2 standard.

7289 **Issue 5**

7290 A table indicating the permissible values of *name* is added to the DESCRIPTION. All those
7291 marked EX are new in this issue.

7292 **Issue 6**

7293 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the
7294 size of the buffer now explicitly states that this includes the terminating null.

7295 The following new requirements on POSIX implementations derive from alignment with the
7296 Single UNIX Specification:

- 7297 • The DESCRIPTION is updated with new arguments which can be used to determine
7298 configuration strings for C compiler flags, linker/loader flags, and libraries for each different
7299 supported programming environment. This is a change to support data size neutrality.

7300 The following changes were made to align with the IEEE P1003.1a draft standard:

- 7301 • The DESCRIPTION is updated to include text describing how `_CS_PATH` can be used to
7302 obtain a *PATH* to access the standard utilities.

7303 The macros associated with the *c89* programming models are marked LEGACY and new
7304 equivalent macros associated with *c99* are introduced.

7305 **NAME**

7306 conj, conjf, conjl — complex conjugate functions

7307 **SYNOPSIS**

7308 #include <complex.h>

7309 double complex conj(double complex z);

7310 float complex conjf(float complex z);

7311 long double complex conjl(long double complex z);

7312 **DESCRIPTION**

7313 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7314 conflict between the requirements described here and the ISO C standard is unintentional. This
7315 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7316 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary
7317 part.

7318 **RETURN VALUE**

7319 These functions return the complex conjugate value.

7320 **ERRORS**

7321 No errors are defined.

7322 **EXAMPLES**

7323 None.

7324 **APPLICATION USAGE**

7325 None.

7326 **RATIONALE**

7327 None.

7328 **FUTURE DIRECTIONS**

7329 None.

7330 **SEE ALSO**

7331 *carg()*, *cimag()*, *cproj()*, *creal()*, the Base Definitions volume of IEEE Std 1003.1-2001,
7332 <complex.h>

7333 **CHANGE HISTORY**

7334 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7335 **NAME**

7336 connect — connect a socket

7337 **SYNOPSIS**

7338 #include <sys/socket.h>

```
7339 int connect(int socket, const struct sockaddr *address,
7340             socklen_t address_len);
```

7341 **DESCRIPTION**

7342 The *connect()* function shall attempt to make a connection on a socket. The function takes the
 7343 following arguments:

7344	<i>socket</i>	Specifies the file descriptor associated with the socket.
7345	<i>address</i>	Points to a sockaddr structure containing the peer address. The length and format of the address depend on the address family of the socket.
7347	<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> argument.

7349 If the socket has not already been bound to a local address, *connect()* shall bind it to an address
 7350 which, unless the socket's address family is AF_UNIX, is an unused local address.

7351 If the initiating socket is not connection-mode, then *connect()* shall set the socket's peer address,
 7352 and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all
 7353 datagrams are sent on subsequent *send()* functions, and limits the remote sender for subsequent
 7354 *recv()* functions. If *address* is a null address for the protocol, the socket's peer address shall be
 7355 reset.

7356 If the initiating socket is connection-mode, then *connect()* shall attempt to establish a connection
 7357 to the address specified by the *address* argument. If the connection cannot be established
 7358 immediately and O_NONBLOCK is not set for the file descriptor for the socket, *connect()* shall
 7359 block for up to an unspecified timeout interval until the connection is established. If the timeout
 7360 interval expires before the connection is established, *connect()* shall fail and the connection
 7361 attempt shall be aborted. If *connect()* is interrupted by a signal that is caught while blocked
 7362 waiting to establish a connection, *connect()* shall fail and set *errno* to [EINTR], but the connection
 7363 request shall not be aborted, and the connection shall be established asynchronously.

7364 If the connection cannot be established immediately and O_NONBLOCK is set for the file
 7365 descriptor for the socket, *connect()* shall fail and set *errno* to [EINPROGRESS], but the connection
 7366 request shall not be aborted, and the connection shall be established asynchronously.
 7367 Subsequent calls to *connect()* for the same socket, before the connection is established, shall fail
 7368 and set *errno* to [EALREADY].

7369 When the connection has been established asynchronously, *select()* and *poll()* shall indicate that
 7370 the file descriptor for the socket is ready for writing.

7371 The socket in use may require the process to have appropriate privileges to use the *connect()*
 7372 function.

7373 **RETURN VALUE**

7374 Upon successful completion, *connect()* shall return 0; otherwise, -1 shall be returned and *errno*
 7375 set to indicate the error.

7376 **ERRORS**

7377 The *connect()* function shall fail if:

7378 [EADDRNOTAVAIL]

7379 The specified address is not available from the local machine.

7380	[EAFNOSUPPORT]	
7381		The specified address is not a valid address for the address family of the
7382		specified socket.
7383	[EALREADY]	A connection request is already in progress for the specified socket.
7384	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
7385	[ECONNREFUSED]	
7386		The target address was not listening for connections or refused the connection
7387		request.
7388	[EINPROGRESS]	O_NONBLOCK is set for the file descriptor for the socket and the connection
7389		cannot be immediately established; the connection shall be established
7390		asynchronously.
7391	[EINTR]	The attempt to establish a connection was interrupted by delivery of a signal
7392		that was caught; the connection shall be established asynchronously.
7393	[EISCONN]	The specified socket is connection-mode and is already connected.
7394	[ENETUNREACH]	
7395		No route to the network is present.
7396	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
7397	[EPROTOTYPE]	The specified address has a different type than the socket bound to the
7398		specified peer address.
7399	[ETIMEDOUT]	The attempt to connect timed out before a connection was made.
7400	If the address family of the socket is AF_UNIX, then <i>connect()</i> shall fail if:	
7401	[EIO]	An I/O error occurred while reading from or writing to the file system.
7402	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname
7403		in <i>address</i> .
7404	[ENAMETOOLONG]	
7405		A component of a pathname exceeded {NAME_MAX} characters, or an entire
7406		pathname exceeded {PATH_MAX} characters.
7407	[ENOENT]	A component of the pathname does not name an existing file or the pathname
7408		is an empty string.
7409	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> is not a directory.
7410	The <i>connect()</i> function may fail if:	
7411	[EACCES]	Search permission is denied for a component of the path prefix; or write
7412		access to the named socket is denied.
7413	[EADDRINUSE]	Attempt to establish a connection that uses addresses that are already in use.
7414	[ECONNRESET]	Remote host reset the connection request.
7415	[EHOSTUNREACH]	
7416		The destination host cannot be reached (probably because the host is down or
7417		a remote router cannot reach it).
7418	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family; or
7419		invalid address family in the sockaddr structure.

7420 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
7421 resolution of the pathname in *address*.

7422 [ENAMETOOLONG]
7423 Pathname resolution of a symbolic link produced an intermediate result
7424 whose length exceeds {PATH_MAX}.

7425 [ENETDOWN] The local network interface used to reach the destination is down.

7426 [ENOBUFS] No buffer space is available.

7427 [EOPNOTSUPP] The socket is listening and cannot be connected.

7428 **EXAMPLES**
7429 None.

7430 **APPLICATION USAGE**
7431 If *connect()* fails, the state of the socket is unspecified. Conforming applications should close the
7432 file descriptor and create a new socket before attempting to reconnect.

7433 **RATIONALE**
7434 None.

7435 **FUTURE DIRECTIONS**
7436 None.

7437 **SEE ALSO**
7438 *accept()*, *bind()*, *close()*, *getsockname()*, *poll()*, *select()*, *send()*, *shutdown()*, *socket()*, the Base
7439 Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>

7440 **CHANGE HISTORY**
7441 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7442 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
7443 [ELOOP] error condition is added.

7444 **NAME**

7445 copysign, copysignf, copysignl — number manipulation function

7446 **SYNOPSIS**

7447 #include <math.h>

7448 double copysign(double x, double y);

7449 float copysignf(float x, float y);

7450 long double copysignl(long double x, long double y);

7451 **DESCRIPTION**

7452 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7453 conflict between the requirements described here and the ISO C standard is unintentional. This
7454 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7455 These functions shall produce a value with the magnitude of *x* and the sign of *y*. On
7456 implementations that represent a signed zero but do not treat negative zero consistently in
7457 arithmetic operations, these functions regard the sign of zero as positive.

7458 **RETURN VALUE**

7459 Upon successful completion, these functions shall return a value with the magnitude of *x* and
7460 the sign of *y*.

7461 **ERRORS**

7462 No errors are defined.

7463 **EXAMPLES**

7464 None.

7465 **APPLICATION USAGE**

7466 None.

7467 **RATIONALE**

7468 None.

7469 **FUTURE DIRECTIONS**

7470 None.

7471 **SEE ALSO**7472 *signbit()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>7473 **CHANGE HISTORY**

7474 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7475 **NAME**

7476 cos, cosf, cosl — cosine function

7477 **SYNOPSIS**

7478 #include <math.h>

7479 double cos(double x);

7480 float cosf(float x);

7481 long double cosl(long double x);

7482 **DESCRIPTION**

7483 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 7484 conflict between the requirements described here and the ISO C standard is unintentional. This
 7485 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7486 These functions shall compute the cosine of their argument x , measured in radians.

7487 An application wishing to check for error situations should set *errno* to zero and call
 7488 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 7489 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 7490 zero, an error has occurred.

7491 **RETURN VALUE**7492 Upon successful completion, these functions shall return the cosine of x .7493 MX If x is NaN, a NaN shall be returned.7494 If x is ± 0 , the value 1.0 shall be returned.

7495 If x is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 7496 defined value shall be returned.

7497 **ERRORS**

7498 These functions shall fail if:

7499 MX **Domain Error** The x argument is $\pm \text{Inf}$.

7500 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 7501 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 7502 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 7503 shall be raised.

7504 **EXAMPLES**7505 **Taking the Cosine of a 45-Degree Angle**

7506 #include <math.h>

7507 ...

7508 double radians = 45 * M_PI / 180;

7509 double result;

7510 ...

7511 result = cos(radians);

7512 **APPLICATION USAGE**

7513 These functions may lose accuracy when their argument is near an odd multiple of $\pi/2$ or is far
 7514 from 0.

7515 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 7516 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

7517 **RATIONALE**

7518 None.

7519 **FUTURE DIRECTIONS**

7520 None.

7521 **SEE ALSO**

7522 *acos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, *tan()*, the Base Definitions volume of
7523 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
7524 **<math.h>**

7525 **CHANGE HISTORY**

7526 First released in Issue 1. Derived from Issue 1 of the SVID.

7527 **Issue 5**

7528 The DESCRIPTION is updated to indicate how an application should check for an error. This
7529 text was previously published in the APPLICATION USAGE section.

7530 **Issue 6**7531 The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

7532 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
7533 revised to align with the ISO/IEC 9899:1999 standard.

7534 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
7535 marked.

7536 **NAME**

7537 cosh, coshf, coshl — hyperbolic cosine functions

7538 **SYNOPSIS**

7539 #include <math.h>

7540 double cosh(double x);

7541 float coshf(float x);

7542 long double coshl(long double x);

7543 **DESCRIPTION**

7544 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 7545 conflict between the requirements described here and the ISO C standard is unintentional. This
 7546 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7547 These functions shall compute the hyperbolic cosine of their argument x .

7548 An application wishing to check for error situations should set *errno* to zero and call
 7549 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 7550 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 7551 zero, an error has occurred.

7552 **RETURN VALUE**7553 Upon successful completion, these functions shall return the hyperbolic cosine of x .

7554 If the correct value would cause overflow, a range error shall occur and *cosh()*, *coshf()*, and
 7555 *coshl()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 7556 respectively.

7557 MX If x is NaN, a NaN shall be returned.7558 If x is ± 0 , the value 1.0 shall be returned.7559 If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.7560 **ERRORS**

7561 These functions shall fail if:

7562 Range Error The result would cause an overflow.

7563 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 7564 then *errno* shall be set to [ERANGE]. If the integer expression
 7565 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 7566 floating-point exception shall be raised.

7567 **EXAMPLES**

7568 None.

7569 **APPLICATION USAGE**

7570 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 7571 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

7572 For IEEE Std 754-1985 **double**, $710.5 < |x|$ implies that *cosh*(x) has overflowed.7573 **RATIONALE**

7574 None.

7575 **FUTURE DIRECTIONS**

7576 None.

7577 **SEE ALSO**

7578 *acosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sinh()*, *tanh()*, the Base Definitions volume of
7579 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
7580 **<math.h>**

7581 **CHANGE HISTORY**

7582 First released in Issue 1. Derived from Issue 1 of the SVID.

7583 **Issue 5**

7584 The DESCRIPTION is updated to indicate how an application should check for an error. This
7585 text was previously published in the APPLICATION USAGE section.

7586 **Issue 6**

7587 The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

7588 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
7589 revised to align with the ISO/IEC 9899:1999 standard.

7590 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
7591 marked.

7592 **NAME**

7593 cosl — cosine function

7594 **SYNOPSIS**

7595 #include <math.h>

7596 long double cosl(long double x);

7597 **DESCRIPTION**7598 Refer to *cos()*.

7599 **NAME**

7600 cpow, cpowf, cpowl — complex power functions

7601 **SYNOPSIS**

7602 #include <complex.h>

7603 double complex cpow(double complex x, double complex y);

7604 float complex cpowf(float complex x, float complex y);

7605 long double complex cpowl(long double complex x,

7606 long double complex y);

7607 **DESCRIPTION**

7608 CX The functionality described on this reference page is aligned with the ISO C standard. Any
7609 conflict between the requirements described here and the ISO C standard is unintentional. This
7610 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7611 These functions shall compute the complex power function x^y , with a branch cut for the first
7612 parameter along the negative real axis.

7613 **RETURN VALUE**

7614 These functions shall return the complex power function value.

7615 **ERRORS**

7616 No errors are defined.

7617 **EXAMPLES**

7618 None.

7619 **APPLICATION USAGE**

7620 None.

7621 **RATIONALE**

7622 None.

7623 **FUTURE DIRECTIONS**

7624 None.

7625 **SEE ALSO**7626 *cabs()*, *csqrt()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7627 **CHANGE HISTORY**

7628 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7629 **NAME**

7630 cproj, cprojf, cprojl — complex projection functions

7631 **SYNOPSIS**

7632 #include <complex.h>

7633 double complex cproj(double complex z);

7634 float complex cprojf(float complex z);

7635 long double complex cprojl(long double complex z);

7636 **DESCRIPTION**

7637 CX The functionality described on this reference page is aligned with the ISO C standard. Any
7638 conflict between the requirements described here and the ISO C standard is unintentional. This
7639 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7640 These functions shall compute a projection of *z* onto the Riemann sphere: *z* projects to *z*, except
7641 that all complex infinities (even those with one infinite part and one NaN part) project to
7642 positive infinity on the real axis. If *z* has an infinite part, then *cproj(z)* shall be equivalent to:

7643 INFINITY + I * copysign(0.0, cimag(z))

7644 **RETURN VALUE**

7645 These functions shall return the value of the projection onto the Riemann sphere.

7646 **ERRORS**

7647 No errors are defined.

7648 **EXAMPLES**

7649 None.

7650 **APPLICATION USAGE**

7651 None.

7652 **RATIONALE**

7653 Two topologies are commonly used in complex mathematics: the complex plane with its
7654 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is
7655 better suited for transcendental functions, the Riemann sphere for algebraic functions. The
7656 complex types with their multiplicity of infinities provide a useful (though imperfect) model for
7657 the complex plane. The *cproj()* function helps model the Riemann sphere by mapping all
7658 infinities to one, and should be used just before any operation, especially comparisons, that
7659 might give spurious results for any of the other infinities. Note that a complex value with one
7660 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is
7661 infinite, the complex value is infinite independent of the value of the other part. For the same
7662 reason, *cabs()* returns an infinity if its argument has an infinite part and a NaN part.

7663 **FUTURE DIRECTIONS**

7664 None.

7665 **SEE ALSO**7666 *carg()*, *cimag()*, *conj()*, *creal()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7667 **CHANGE HISTORY**

7668 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7669 **NAME**7670 `creal, crealf, creall` — complex real functions7671 **SYNOPSIS**7672 `#include <complex.h>`7673 `double creal(double complex z);`7674 `float crealf(float complex z);`7675 `long double creall(long double complex z);`7676 **DESCRIPTION**

7677 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7678 conflict between the requirements described here and the ISO C standard is unintentional. This
7679 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7680 These functions shall compute the real part of *z*.7681 **RETURN VALUE**

7682 These functions shall return the real part value.

7683 **ERRORS**

7684 No errors are defined.

7685 **EXAMPLES**

7686 None.

7687 **APPLICATION USAGE**7688 For a variable *z* of type **complex**:7689 `z == creal(z) + cimag(z)*I`7690 **RATIONALE**

7691 None.

7692 **FUTURE DIRECTIONS**

7693 None.

7694 **SEE ALSO**

7695 `carg()`, `cimag()`, `conj()`, `cproj()`, the Base Definitions volume of IEEE Std 1003.1-2001,
7696 **<complex.h>**

7697 **CHANGE HISTORY**

7698 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7699 **NAME**

7700 creat — create a new file or rewrite an existing one

7701 **SYNOPSIS**

7702 OH #include <sys/stat.h>

7703 #include <fcntl.h>

7704 int creat(const char *path, mode_t mode);

7705 **DESCRIPTION**

7706 The function call:

7707 creat(path, mode)

7708 shall be equivalent to:

7709 open(path, O_WRONLY|O_CREAT|O_TRUNC, mode)

7710 **RETURN VALUE**7711 Refer to *open()*.7712 **ERRORS**7713 Refer to *open()*.7714 **EXAMPLES**7715 **Creating a File**

7716 The following example creates the file **/tmp/file** with read and write permissions for the file
 7717 owner and read permission for group and others. The resulting file descriptor is assigned to the
 7718 *fd* variable.

7719 #include <fcntl.h>

7720 ...

7721 int fd;

7722 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;

7723 char *filename = "/tmp/file";

7724 ...

7725 fd = creat(filename, mode);

7726 ...

7727 **APPLICATION USAGE**

7728 None.

7729 **RATIONALE**

7730 The *creat()* function is redundant. Its services are also provided by the *open()* function. It has
 7731 been included primarily for historical purposes since many existing applications depend on it. It
 7732 is best considered a part of the C binding rather than a function that should be provided in other
 7733 languages.

7734 **FUTURE DIRECTIONS**

7735 None.

7736 **SEE ALSO**

7737 *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <fcntl.h>, <sys/stat.h>,
 7738 <sys/types.h>

7739 **CHANGE HISTORY**

7740 First released in Issue 1. Derived from Issue 1 of the SVID.

7741 **Issue 6**7742 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.7743 The following new requirements on POSIX implementations derive from alignment with the
7744 Single UNIX Specification:

- 7745 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
7746 required for conforming implementations of previous POSIX specifications, it was not
7747 required for UNIX applications.

7748 **NAME**7749 crypt — string encoding function (**CRYPT**)7750 **SYNOPSIS**7751 xSI `#include <unistd.h>`7752 `char *crypt(const char *key, const char *salt);`

7753

7754 **DESCRIPTION**7755 The *crypt()* function is a string encoding function. The algorithm is implementation-defined.7756 The *key* argument points to a string to be encoded. The *salt* argument is a string chosen from the
7757 set:

7758 a b c d e f g h i j k l m n o p q r s t u v w x y z

7759 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

7760 0 1 2 3 4 5 6 7 8 9 . /

7761 The first two characters of this string may be used to perturb the encoding algorithm.

7762 The return value of *crypt()* points to static data that is overwritten by each call.7763 The *crypt()* function need not be reentrant. A function that is not required to be reentrant is not
7764 required to be thread-safe.7765 **RETURN VALUE**7766 Upon successful completion, *crypt()* shall return a pointer to the encoded string. The first two
7767 characters of the returned value shall be those of the *salt* argument. Otherwise, it shall return a
7768 null pointer and set *errno* to indicate the error.7769 **ERRORS**7770 The *crypt()* function shall fail if:

7771 [ENOSYS] The functionality is not supported on this implementation.

7772 **EXAMPLES**7773 **Encoding Passwords**7774 The following example finds a user database entry matching a particular user name and changes
7775 the current password to a new password. The *crypt()* function generates an encoded version of
7776 each password. The first call to *crypt()* produces an encoded version of the old password; that
7777 encoded password is then compared to the password stored in the user database. The second
7778 call to *crypt()* encodes the new password before it is stored.7779 The *putpwent()* function, used in the following example, is not part of IEEE Std 1003.1-2001.7780 `#include <unistd.h>`7781 `#include <pwd.h>`7782 `#include <string.h>`7783 `#include <stdio.h>`7784 `...`7785 `int valid_change;`7786 `int pfd; /* Integer for file descriptor returned by open(). */`7787 `FILE *fpfd; /* File pointer for use in putpwent(). */`7788 `struct passwd *p;`7789 `char user[100];`7790 `char oldpasswd[100];`7791 `char newpasswd[100];`

```

7792     char savepasswd[100];
7793     ...
7794     valid_change = 0;
7795     while ((p = getpwent()) != NULL) {
7796         /* Change entry if found. */
7797         if (strcmp(p->pw_name, user) == 0) {
7798             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
7799                 strcpy(savepasswd, crypt(newpasswd, user));
7800                 p->pw_passwd = savepasswd;
7801                 valid_change = 1;
7802             }
7803             else {
7804                 fprintf(stderr, "Old password is not valid\n");
7805             }
7806         }
7807         /* Put passwd entry into ptmp. */
7808         putpwent(p, fpfd);
7809     }

```

7810 APPLICATION USAGE

7811 The values returned by this function need not be portable among XSI-conformant systems.

7812 RATIONALE

7813 None.

7814 FUTURE DIRECTIONS

7815 None.

7816 SEE ALSO

7817 *encrypt()*, *setkey()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

7818 CHANGE HISTORY

7819 First released in Issue 1. Derived from Issue 1 of the SVID.

7820 Issue 5

7821 Normative text previously in the APPLICATION USAGE section is moved to the
7822 DESCRIPTION.

7823 **NAME**

7824 csin, csinf, csinl — complex sine functions

7825 **SYNOPSIS**

7826 #include <complex.h>

7827 double complex csin(double complex z);

7828 float complex csinf(float complex z);

7829 long double complex csinl(long double complex z);

7830 **DESCRIPTION**

7831 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7832 conflict between the requirements described here and the ISO C standard is unintentional. This
7833 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7834 These functions shall compute the complex sine of *z*.7835 **RETURN VALUE**

7836 These functions shall return the complex sine value.

7837 **ERRORS**

7838 No errors are defined.

7839 **EXAMPLES**

7840 None.

7841 **APPLICATION USAGE**

7842 None.

7843 **RATIONALE**

7844 None.

7845 **FUTURE DIRECTIONS**

7846 None.

7847 **SEE ALSO**7848 `casin()`, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7849 **CHANGE HISTORY**

7850 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7851 **NAME**

7852 csinh, csinhf, csinhl — complex hyperbolic sine functions

7853 **SYNOPSIS**

7854 #include <complex.h>

7855 double complex csinh(double complex z);

7856 float complex csinhf(float complex z);

7857 long double complex csinhl(long double complex z);

7858 **DESCRIPTION**

7859 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7860 conflict between the requirements described here and the ISO C standard is unintentional. This
7861 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7862 These functions shall compute the complex hyperbolic sine of *z*.7863 **RETURN VALUE**

7864 These functions shall return the complex hyperbolic sine value.

7865 **ERRORS**

7866 No errors are defined.

7867 **EXAMPLES**

7868 None.

7869 **APPLICATION USAGE**

7870 None.

7871 **RATIONALE**

7872 None.

7873 **FUTURE DIRECTIONS**

7874 None.

7875 **SEE ALSO**7876 *casinh()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7877 **CHANGE HISTORY**

7878 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7879 **NAME**

7880 csinl — complex sine functions

7881 **SYNOPSIS**

7882 #include <complex.h>

7883 long double complex csinl(long double complex *z*);7884 **DESCRIPTION**7885 Refer to *csin()*.

7886 **NAME**

7887 csqrt, csqrtf, csqrtl — complex square root functions

7888 **SYNOPSIS**

7889 #include <complex.h>

7890 double complex csqrt(double complex z);

7891 float complex csqrtf(float complex z);

7892 long double complex csqrtl(long double complex z);

7893 **DESCRIPTION**

7894 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7895 conflict between the requirements described here and the ISO C standard is unintentional. This
7896 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7897 These functions shall compute the complex square root of *z*, with a branch cut along the
7898 negative real axis.

7899 **RETURN VALUE**

7900 These functions shall return the complex square root value, in the range of the right half-plane
7901 (including the imaginary axis).

7902 **ERRORS**

7903 No errors are defined.

7904 **EXAMPLES**

7905 None.

7906 **APPLICATION USAGE**

7907 None.

7908 **RATIONALE**

7909 None.

7910 **FUTURE DIRECTIONS**

7911 None.

7912 **SEE ALSO**7913 *cabs()*, *cpow()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7914 **CHANGE HISTORY**

7915 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7916 **NAME**

7917 ctan, ctanf, ctanl — complex tangent functions

7918 **SYNOPSIS**

7919 #include <complex.h>

7920 double complex ctan(double complex *z*);7921 float complex ctanf(float complex *z*);7922 long double complex ctanl(long double complex *z*);7923 **DESCRIPTION**

7924 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7925 conflict between the requirements described here and the ISO C standard is unintentional. This
7926 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7927 These functions shall compute the complex tangent of *z*.7928 **RETURN VALUE**

7929 These functions shall return the complex tangent value.

7930 **ERRORS**

7931 No errors are defined.

7932 **EXAMPLES**

7933 None.

7934 **APPLICATION USAGE**

7935 None.

7936 **RATIONALE**

7937 None.

7938 **FUTURE DIRECTIONS**

7939 None.

7940 **SEE ALSO**7941 *catan*(), the Base Definitions volume of IEEE Std 1003.1-2001, <**complex.h**>7942 **CHANGE HISTORY**

7943 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7944 **NAME**

7945 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

7946 **SYNOPSIS**

7947 #include <complex.h>

7948 double complex ctanh(double complex z);

7949 float complex ctanhf(float complex z);

7950 long double complex ctanhl(long double complex z);

7951 **DESCRIPTION**

7952 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7953 conflict between the requirements described here and the ISO C standard is unintentional. This
7954 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7955 These functions shall compute the complex hyperbolic tangent of *z*.7956 **RETURN VALUE**

7957 These functions shall return the complex hyperbolic tangent value.

7958 **ERRORS**

7959 No errors are defined.

7960 **EXAMPLES**

7961 None.

7962 **APPLICATION USAGE**

7963 None.

7964 **RATIONALE**

7965 None.

7966 **FUTURE DIRECTIONS**

7967 None.

7968 **SEE ALSO**7969 *catanh()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7970 **CHANGE HISTORY**

7971 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7972 **NAME**

7973 ctanl — complex tangent functions

7974 **SYNOPSIS**

7975 #include <complex.h>

7976 long double complex ctanl(long double complex *z*);7977 **DESCRIPTION**7978 Refer to *ctan()*.

7979 **NAME**

7980 ctermid — generate a pathname for the controlling terminal

7981 **SYNOPSIS**7982 **CX** #include <stdio.h>

7983 char *ctermid(char *s);

7984

7985 **DESCRIPTION**

7986 The *ctermid()* function shall generate a string that, when used as a pathname, refers to the
 7987 current controlling terminal for the current process. If *ctermid()* returns a pathname, access to the
 7988 file is not guaranteed.

7989 If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`
 7990 functions, it shall ensure that the *ctermid()* function is called with a non-NULL parameter.

7991 **RETURN VALUE**

7992 If *s* is a null pointer, the string shall be generated in an area that may be static (and therefore may
 7993 be overwritten by each call), the address of which shall be returned. Otherwise, *s* is assumed to
 7994 point to a character array of at least `L_ctermid` bytes; the string is placed in this array and the
 7995 value of *s* shall be returned. The symbolic constant `L_ctermid` is defined in <stdio.h>, and shall
 7996 have a value greater than 0.

7997 The *ctermid()* function shall return an empty string if the pathname that would refer to the
 7998 controlling terminal cannot be determined, or if the function is unsuccessful.

7999 **ERRORS**

8000 No errors are defined.

8001 **EXAMPLES**8002 **Determining the Controlling Terminal for the Current Process**

8003 The following example returns a pointer to a string that identifies the controlling terminal for the
 8004 current process. The pathname for the terminal is stored in the array pointed to by the *ptr*
 8005 argument, which has a size of `L_ctermid` bytes, as indicated by the *term* argument.

```
8006       #include <stdio.h>
8007       ...
8008       char term[L_ctermid];
8009       char *ptr;
8010       ptr = ctermid(term);
```

8011 **APPLICATION USAGE**

8012 The difference between *ctermid()* and *ttyname()* is that *ttyname()* must be handed a file
 8013 descriptor and return a path of the terminal associated with that file descriptor, while *ctermid()*
 8014 returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a
 8015 pathname.

8016 **RATIONALE**

8017 `L_ctermid` must be defined appropriately for a given implementation and must be greater than
 8018 zero so that array declarations using it are accepted by the compiler. The value includes the
 8019 terminating null byte.

8020 Conforming applications that use threads cannot call *ctermid()* with NULL as the parameter if
 8021 either `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` is defined. If *s* is not
 8022 NULL, the *ctermid()* function generates a string that, when used as a pathname, refers to the

8023 current controlling terminal for the current process. If *s* is NULL, the return value of *ctermid()* is
8024 undefined.

8025 There is no additional burden on the programmer—changing to use a hypothetical thread-safe
8026 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a
8027 buffer. Application code should not assume that the returned string is short, as some
8028 implementations have more than two pathname components before reaching a logical device
8029 name.

8030 **FUTURE DIRECTIONS**

8031 None.

8032 **SEE ALSO**

8033 *ttyname()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>

8034 **CHANGE HISTORY**

8035 First released in Issue 1. Derived from Issue 1 of the SVID.

8036 **Issue 5**

8037 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8038 **Issue 6**

8039 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8040 **NAME**

8041 ctime, ctime_r — convert a time value to a date and time string

8042 **SYNOPSIS**

8043 #include <time.h>

8044 char *ctime(const time_t *clock);

8045 TSF char *ctime_r(const time_t *clock, char *buf);

8046

8047 **DESCRIPTION**

8048 CX For *ctime()*: The functionality described on this reference page is aligned with the ISO C
 8049 standard. Any conflict between the requirements described here and the ISO C standard is
 8050 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8051 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds
 8052 since the Epoch, to local time in the form of a string. It shall be equivalent to:

8053 asctime(localtime(clock))

8054 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 8055 objects: a broken-down time structure and an array of **char**. Execution of any of the functions
 8056 may overwrite the information returned in either of these objects by any of the other functions.

8057 The *ctime()* function need not be reentrant. A function that is not required to be reentrant is not
 8058 required to be thread-safe.

8059 TSF The *ctime_r()* function shall convert the calendar time pointed to by *clock* to local time in exactly
 8060 the same form as *ctime()* and put the string into the array pointed to by *buf* (which shall be at
 8061 least 26 bytes in size) and return *buf*.

8062 Unlike *ctime()*, the thread-safe version *ctime_r()* is not required to set *tzname*.8063 **RETURN VALUE**

8064 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time
 8065 as an argument.

8066 TSF Upon successful completion, *ctime_r()* shall return a pointer to the string pointed to by *buf*.
 8067 When an error is encountered, a null pointer shall be returned.

8068 **ERRORS**

8069 No errors are defined.

8070 **EXAMPLES**

8071 None.

8072 **APPLICATION USAGE**

8073 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.
 8074 The *ctime()* function is included for compatibility with older implementations, and does not
 8075 support localized date and time formats. Applications should use the *strftime()* function to
 8076 achieve maximum portability.

8077 The *ctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead of
 8078 possibly using a static data area that may be overwritten by each call.

8079 **RATIONALE**

8080 None.

8081 **FUTURE DIRECTIONS**

8082 None.

8083 **SEE ALSO**

8084 *asctime()*, *clock()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
8085 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

8086 **CHANGE HISTORY**

8087 First released in Issue 1. Derived from Issue 1 of the SVID.

8088 **Issue 5**

8089 Normative text previously in the APPLICATION USAGE section is moved to the
8090 DESCRIPTION.

8091 The *ctime_r()* function is included for alignment with the POSIX Threads Extension.8092 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.8093 **Issue 6**

8094 Extensions beyond the ISO C standard are marked.

8095 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8096 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
8097 its avoidance of possibly using a static data area.

8098 **NAME**
8099 daylight — daylight savings time flag

8100 **SYNOPSIS**

8101 xSI `#include <time.h>`
8102 `extern int daylight;`
8103

8104 **DESCRIPTION**

8105 Refer to `tzset()`.

8106 **NAME**

8107 dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey,
8108 dbm_open, dbm_store — database functions

8109 **SYNOPSIS**

```
8110 xSI    #include <ndbm.h>

8111    int dbm_clearerr(DBM *db);
8112    void dbm_close(DBM *db);
8113    int dbm_delete(DBM *db, datum key);
8114    int dbm_error(DBM *db);
8115    datum dbm_fetch(DBM *db, datum key);
8116    datum dbm_firstkey(DBM *db);
8117    datum dbm_nextkey(DBM *db);
8118    DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
8119    int dbm_store(DBM *db, datum key, datum content, int store_mode);
8120
```

8121 **DESCRIPTION**

8122 These functions create, access, and modify a database.

8123 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object
8124 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in
8125 the object pointed to by *dptr*.

8126 The database is stored in two files. One file is a directory containing a bitmap of keys and has
8127 **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

8128 The *dbm_open()* function shall open a database. The *file* argument to the function is the
8129 pathname of the database. The function opens two files named *file.dir* and *file.pag*. The
8130 *open_flags* argument has the same meaning as the *flags* argument of *open()* except that a database
8131 opened for write-only access opens the files for read and write access and the behavior of the
8132 O_APPEND flag is unspecified. The *file_mode* argument has the same meaning as the third
8133 argument of *open()*.

8134 The *dbm_close()* function shall close a database. The application shall ensure that argument *db* is
8135 a pointer to a **dbm** structure that has been returned from a call to *dbm_open()*.

8136 These database functions shall support an internal block size large enough to support
8137 key/content pairs of at least 1023 bytes.

8138 The *dbm_fetch()* function shall read a record from a database. The argument *db* is a pointer to a
8139 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
8140 **datum** that has been initialized by the application to the value of the key that matches the key of
8141 the record the program is fetching.

8142 The *dbm_store()* function shall write a record to a database. The argument *db* is a pointer to a
8143 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
8144 **datum** that has been initialized by the application to the value of the key that identifies (for
8145 subsequent reading, writing, or deleting) the record the application is writing. The argument
8146 *content* is a **datum** that has been initialized by the application to the value of the record the
8147 program is writing. The argument *store_mode* controls whether *dbm_store()* replaces any pre-
8148 existing record that has the same key that is specified by the *key* argument. The application shall
8149 set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that
8150 matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record shall be
8151 replaced with the new record. If the database contains a record that matches the *key* argument
8152 and *store_mode* is DBM_INSERT, the existing record shall be left unchanged and the new record

8153 ignored. If the database does not contain a record that matches the *key* argument and *store_mode*
 8154 is either DBM_INSERT or DBM_REPLACE, the new record shall be inserted in the database.

8155 If the sum of a key/content pair exceeds the internal block size, the result is unspecified.
 8156 Moreover, the application shall ensure that all key/content pairs that hash together fit on a
 8157 single block. The *dbm_store()* function shall return an error in the event that a disk block fills
 8158 with inseparable data.

8159 The *dbm_delete()* function shall delete a record and its key from the database. The argument *db* is
 8160 a pointer to a database structure that has been returned from a call to *dbm_open()*. The argument
 8161 *key* is a **datum** that has been initialized by the application to the value of the key that identifies
 8162 the record the program is deleting.

8163 The *dbm_firstkey()* function shall return the first key in the database. The argument *db* is a
 8164 pointer to a database structure that has been returned from a call to *dbm_open()*.

8165 The *dbm_nextkey()* function shall return the next key in the database. The argument *db* is a
 8166 pointer to a database structure that has been returned from a call to *dbm_open()*. The application
 8167 shall ensure that the *dbm_firstkey()* function is called before calling *dbm_nextkey()*. Subsequent
 8168 calls to *dbm_nextkey()* return the next key until all of the keys in the database have been
 8169 returned.

8170 The *dbm_error()* function shall return the error condition of the database. The argument *db* is a
 8171 pointer to a database structure that has been returned from a call to *dbm_open()*.

8172 The *dbm_clearerr()* function shall clear the error condition of the database. The argument *db* is a
 8173 pointer to a database structure that has been returned from a call to *dbm_open()*.

8174 The *dptr* pointers returned by these functions may point into static storage that may be changed
 8175 by subsequent calls.

8176 These functions need not be reentrant. A function that is not required to be reentrant is not
 8177 required to be thread-safe.

8178 **RETURN VALUE**

8179 The *dbm_store()* and *dbm_delete()* functions shall return 0 when they succeed and a negative
 8180 value when they fail.

8181 The *dbm_store()* function shall return 1 if it is called with a *flags* value of DBM_INSERT and the
 8182 function finds an existing record with the same key.

8183 The *dbm_error()* function shall return 0 if the error condition is not set and return a non-zero
 8184 value if the error condition is set.

8185 The return value of *dbm_clearerr()* is unspecified.

8186 The *dbm_firstkey()* and *dbm_nextkey()* functions shall return a key **datum**. When the end of the
 8187 database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr*
 8188 member of the key shall be a null pointer and the error condition of the database shall be set.

8189 The *dbm_fetch()* function shall return a content **datum**. If no record in the database matches the
 8190 key or if an error condition has been detected in the database, the *dptr* member of the content
 8191 shall be a null pointer.

8192 The *dbm_open()* function shall return a pointer to a database structure. If an error is detected
 8193 during the operation, *dbm_open()* shall return a (**DBM ***)0.

8194 **ERRORS**

8195 No errors are defined.

8196 **EXAMPLES**

8197 None.

8198 **APPLICATION USAGE**

8199 The following code can be used to traverse the database:

8200 `for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))`

8201 The *dbm_** functions provided in this library should not be confused in any way with those of a
 8202 general-purpose database management system. These functions do not provide for multiple
 8203 search keys per entry, they do not protect against multi-user access (in other words they do not
 8204 lock records or files), and they do not provide the many other useful database functions that are
 8205 found in more robust database management systems. Creating and updating databases by use of
 8206 these functions is relatively slow because of data copies that occur upon hash collisions. These
 8207 functions are useful for applications requiring fast lookup of relatively static information that is
 8208 to be indexed by a single key.

8209 Note that a strictly conforming application is extremely limited by these functions: since there is
 8210 no way to determine that the keys in use do not all hash to the same value (although that would
 8211 be rare), a strictly conforming application cannot be guaranteed that it can store more than one
 8212 block's worth of data in the database. As long as a key collision does not occur, additional data
 8213 may be stored, but because there is no way to determine whether an error is due to a key
 8214 collision or some other error condition (*dbm_error()* being effectively a Boolean), once an error is
 8215 detected, the application is effectively limited to guessing what the error might be if it wishes to
 8216 continue using these functions.

8217 The *dbm_delete()* function need not physically reclaim file space, although it does make it
 8218 available for reuse by the database.

8219 After calling *dbm_store()* or *dbm_delete()* during a pass through the keys by *dbm_firstkey()* and
 8220 *dbm_nextkey()*, the application should reset the database by calling *dbm_firstkey()* before again
 8221 calling *dbm_nextkey()*. The contents of these files are unspecified and may not be portable.

8222 **RATIONALE**

8223 None.

8224 **FUTURE DIRECTIONS**

8225 None.

8226 **SEE ALSO**8227 *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**ndbm.h**>8228 **CHANGE HISTORY**

8229 First released in Issue 4, Version 2.

8230 **Issue 5**

8231 Moved from X/OPEN UNIX extension to BASE.

8232 Normative text previously in the APPLICATION USAGE section is moved to the
 8233 DESCRIPTION.

8234 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8235 **Issue 6**

8236 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8237 NAME

8238 difftime — compute the difference between two calendar time values

8239 SYNOPSIS

8240 #include <time.h>

8241 double difftime(time_t *time1*, time_t *time0*);

8242 DESCRIPTION

8243 cx The functionality described on this reference page is aligned with the ISO C standard. Any
8244 conflict between the requirements described here and the ISO C standard is unintentional. This
8245 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8246 The *difftime()* function shall compute the difference between two calendar times (as returned by
8247 *time()*): *time1* − *time0*.

8248 RETURN VALUE

8249 The *difftime()* function shall return the difference expressed in seconds as a type **double**.

8250 ERRORS

8251 No errors are defined.

8252 EXAMPLES

8253 None.

8254 APPLICATION USAGE

8255 None.

8256 RATIONALE

8257 None.

8258 FUTURE DIRECTIONS

8259 None.

8260 SEE ALSO

8261 *asctime()*, *clock()*, *ctime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
8262 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

8263 CHANGE HISTORY

8264 First released in Issue 4. Derived from the ISO C standard.

8265 **NAME**

8266 dirname — report the parent directory name of a file pathname

8267 **SYNOPSIS**

8268 xSI #include <libgen.h>

8269 char *dirname(char *path);

8270

8271 **DESCRIPTION**

8272 The *dirname()* function shall take a pointer to a character string that contains a pathname, and
 8273 return a pointer to a string that is a pathname of the parent directory of that file. Trailing '/'
 8274 characters in the path are not counted as part of the path.

8275 If *path* does not contain a '/', then *dirname()* shall return a pointer to the string ".". If *path* is a
 8276 null pointer or points to an empty string, *dirname()* shall return a pointer to the string ".".

8277 The *dirname()* function need not be reentrant. A function that is not required to be reentrant is
 8278 not required to be thread-safe.

8279 **RETURN VALUE**

8280 The *dirname()* function shall return a pointer to a string that is the parent directory of *path*. If
 8281 *path* is a null pointer or points to an empty string, a pointer to a string "." is returned.

8282 The *dirname()* function may modify the string pointed to by *path*, and may return a pointer to
 8283 static storage that may then be overwritten by subsequent calls to *dirname()*.

8284 **ERRORS**

8285 No errors are defined.

8286 **EXAMPLES**

8287 The following code fragment reads a pathname, changes the current working directory to the
 8288 parent directory, and opens the file.

```
8289 char path[PATH_MAX], *pathcopy;
8290 int fd;
8291 fgets(path, PATH_MAX, stdin);
8292 pathcopy = strdup(path);
8293 chdir(dirname(pathcopy));
8294 fd = open(basename(path), O_RDONLY);
```

8295 **Sample Input and Output Strings for dirname()**

8296 In the following table, the input string is the value pointed to by *path*, and the output string is
 8297 the return value of the *dirname()* function.

Input String	Output String
"/usr/lib"	"/usr"
"/usr/"	"/"
"usr"	"."
"/"	"/"
"."	"."
".."	"."

Changing the Current Directory to the Parent Directory

The following program fragment reads a pathname, changes the current working directory to the parent directory, and opens the file.

```
#include <unistd.h>
#include <limits.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <libgen.h>
...
char path[PATH_MAX], *pathcopy;
int fd;
...
fgets(path, PATH_MAX, stdin);
pathcopy = strdup(path);
chdir(dirname(pathcopy));
fd = open(basename(path), O_RDONLY);
```

APPLICATION USAGE

The *dirname()* and *basename()* functions together yield a complete pathname. The expression *dirname(path)* obtains the pathname of the directory where *basename(path)* is found.

Since the meaning of the leading *"/"* is implementation-defined, *dirname("//foo)* may return either *"/"* or *'/'* (but nothing else).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

basename(), the Base Definitions volume of IEEE Std 1003.1-2001, **<libgen.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

8340 **NAME**8341 `div` — compute the quotient and remainder of an integer division8342 **SYNOPSIS**8343 `#include <stdlib.h>`8344 `div_t div(int numer, int denom);`8345 **DESCRIPTION**

8346 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
8347 conflict between the requirements described here and the ISO C standard is unintentional. This
8348 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8349 The `div()` function shall compute the quotient and remainder of the division of the numerator
8350 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer
8351 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be
8352 represented, the behavior is undefined; otherwise, *quot*denom+rem* shall equal *numer*.

8353 **RETURN VALUE**

8354 The `div()` function shall return a structure of type **div_t**, comprising both the quotient and the
8355 remainder. The structure includes the following members, in any order:

8356 `int quot; /* quotient */`8357 `int rem; /* remainder */`8358 **ERRORS**

8359 No errors are defined.

8360 **EXAMPLES**

8361 None.

8362 **APPLICATION USAGE**

8363 None.

8364 **RATIONALE**

8365 None.

8366 **FUTURE DIRECTIONS**

8367 None.

8368 **SEE ALSO**8369 `ldiv()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`8370 **CHANGE HISTORY**

8371 First released in Issue 4. Derived from the ISO C standard.

8372 **NAME**8373 dldclose — close a *dlopen()* object8374 **SYNOPSIS**

8375 xSI #include <dldfcn.h>

8376 int dldclose(void *handle);

8377

8378 **DESCRIPTION**8379 The *dldclose()* function shall inform the system that the object referenced by a *handle* returned
8380 from a previous *dlopen()* invocation is no longer needed by the application.8381 The use of *dldclose()* reflects a statement of intent on the part of the process, but does not create
8382 any requirement upon the implementation, such as removal of the code or symbols referenced
8383 by *handle*. Once an object has been closed using *dldclose()* an application should assume that its
8384 symbols are no longer available to *dlsym()*. All objects loaded automatically as a result of
8385 invoking *dlopen()* on the referenced object shall also be closed if this is the last reference to it.8386 Although a *dldclose()* operation is not required to remove structures from an address space,
8387 neither is an implementation prohibited from doing so. The only restriction on such a removal is
8388 that no object shall be removed to which references have been relocated, until or unless all such
8389 references are removed. For instance, an object that had been loaded with a *dlopen()* operation
8390 specifying the RTLD_GLOBAL flag might provide a target for dynamic relocations performed in
8391 the processing of other objects—in such environments, an application may assume that no
8392 relocation, once made, shall be undone or remade unless the object requiring the relocation has
8393 itself been removed.8394 **RETURN VALUE**8395 If the referenced object was successfully closed, *dldclose()* shall return 0. If the object could not be
8396 closed, or if *handle* does not refer to an open object, *dldclose()* shall return a non-zero value. More
8397 detailed diagnostic information shall be available through *dlderror()*.8398 **ERRORS**

8399 No errors are defined.

8400 **EXAMPLES**8401 The following example illustrates use of *dlopen()* and *dldclose()*:8402 ...
8403 /* Open a dynamic library and then close it ... */
8404 #include <dldfcn.h>
8405 void *mylib;
8406 int eret;
8407 mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
8408 ...
8409 eret = dldclose(mylib);
8410 ...8411 **APPLICATION USAGE**8412 A conforming application should employ a *handle* returned from a *dlopen()* invocation only
8413 within a given scope bracketed by the *dlopen()* and *dldclose()* operations. Implementations are
8414 free to use reference counting or other techniques such that multiple calls to *dlopen()* referencing
8415 the same object may return the same object for *handle*. Implementations are also free to reuse a
8416 *handle*. For these reasons, the value of a *handle* must be treated as an opaque object by the
8417 application, used only in calls to *dlsym()* and *dldclose()*.

8418 **RATIONALE**

8419 None.

8420 **FUTURE DIRECTIONS**

8421 None.

8422 **SEE ALSO**8423 *dLError()*, *dlopen()*, *dlsym()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**dlfcn.h**>8424 **CHANGE HISTORY**

8425 First released in Issue 5.

8426 **Issue 6**8427 The DESCRIPTION is updated to say that the referenced object is closed “if this is the last
8428 reference to it”.

8429 **NAME**8430 `dlderror` — get diagnostic information8431 **SYNOPSIS**8432 `XS1 #include <dldfcn.h>`8433 `char *dlderror(void);`

8434

8435 **DESCRIPTION**

8436 The `dlderror()` function shall return a null-terminated character string (with no trailing <newline>) that describes the last error that occurred during dynamic linking processing. If no dynamic linking errors have occurred since the last invocation of `dlderror()`, `dlderror()` shall return NULL. Thus, invoking `dlderror()` a second time, immediately following a prior invocation, shall result in NULL being returned.

8441 The `dlderror()` function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

8443 **RETURN VALUE**

8444 If successful, `dlderror()` shall return a null-terminated character string; otherwise, NULL shall be returned.

8446 **ERRORS**

8447 No errors are defined.

8448 **EXAMPLES**

8449 The following example prints out the last dynamic linking error:

```
8450 ...
8451 #include <dldfcn.h>
8452 char *errstr;
8453 errstr = dlderror();
8454 if (errstr != NULL)
8455     printf ("A dynamic linking error occurred: (%s)\n", errstr);
8456 ...
```

8457 **APPLICATION USAGE**

8458 The messages returned by `dlderror()` may reside in a static buffer that is overwritten on each call to `dlderror()`. Application code should not write to this buffer. Programs wishing to preserve an error message should make their own copies of that message. Depending on the application environment with respect to asynchronous execution events, such as signals or other asynchronous computation sharing the address space, conforming applications should use a critical section to retrieve the error pointer and buffer.

8464 **RATIONALE**

8465 None.

8466 **FUTURE DIRECTIONS**

8467 None.

8468 **SEE ALSO**8469 `dldclose()`, `dldopen()`, `dldsym()`, the Base Definitions volume of IEEE Std 1003.1-2001, <dldfcn.h>

8470 **CHANGE HISTORY**

8471 First released in Issue 5.

8472 **Issue 6**

8473 In the DESCRIPTION the note about reentrancy and thread-safety is added.

8474 **NAME**

8475 dlopen — gain access to an executable object file

8476 **SYNOPSIS**

8477 xSI #include <dlfcn.h>

8478 void *dlopen(const char *file, int mode);

8479

8480 **DESCRIPTION**

8481 The *dlopen()* function shall make an executable object file specified by *file* available to the calling
 8482 program. The class of files eligible for this operation and the manner of their construction are
 8483 implementation-defined, though typically such files are executable objects such as shared
 8484 libraries, relocatable files, or programs. Note that some implementations permit the construction
 8485 of dependencies between such objects that are embedded within files. In such cases, a *dlopen()*
 8486 operation shall load such dependencies in addition to the object referenced by *file*.
 8487 Implementations may also impose specific constraints on the construction of programs that can
 8488 employ *dlopen()* and its related services.

8489 A successful *dlopen()* shall return a *handle* which the caller may use on subsequent calls to
 8490 *dlsym()* and *dlclose()*. The value of this *handle* should not be interpreted in any way by the caller.

8491 The *file* argument is used to construct a pathname to the object file. If *file* contains a slash
 8492 character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in an
 8493 implementation-defined manner to yield a pathname.

8494 If the value of *file* is 0, *dlopen()* shall provide a *handle* on a global symbol object. This object shall
 8495 provide access to the symbols from an ordered set of objects consisting of the original program
 8496 image file, together with any objects loaded at program start-up as specified by that process
 8497 image file (for example, shared libraries), and the set of objects loaded using a *dlopen()* operation
 8498 together with the RTLD_GLOBAL flag. As the latter set of objects can change during execution,
 8499 the set identified by *handle* can also change dynamically.

8500 Only a single copy of an object file is brought into the address space, even if *dlopen()* is invoked
 8501 multiple times in reference to the file, and even if different pathnames are used to reference the
 8502 file.

8503 The *mode* parameter describes how *dlopen()* shall operate upon *file* with respect to the processing
 8504 of relocations and the scope of visibility of the symbols provided within *file*. When an object is
 8505 brought into the address space of a process, it may contain references to symbols whose
 8506 addresses are not known until the object is loaded. These references shall be relocated before the
 8507 symbols can be accessed. The *mode* parameter governs when these relocations take place and
 8508 may have the following values:

8509	RTLD_LAZY	Relocations shall be performed at an implementation-defined time,
8510		ranging from the time of the <i>dlopen()</i> call until the first reference to a
8511		given symbol occurs. Specifying RTLD_LAZY should improve
8512		performance on implementations supporting dynamic symbol binding as
8513		a process may not reference all of the functions in any given object. And,
8514		for systems supporting dynamic symbol resolution for normal process
8515		execution, this behavior mimics the normal handling of process
8516		execution.

8517	RTLD_NOW	All necessary relocations shall be performed when the object is first
8518		loaded. This may waste some processing if relocations are performed for
8519		functions that are never referenced. This behavior may be useful for
8520		applications that need to know as soon as an object is loaded that all

8521 symbols referenced during execution are available.

8522 Any object loaded by *dlopen()* that requires relocations against global symbols can reference the
 8523 symbols in the original process image file, any objects loaded at program start-up, from the
 8524 object itself as well as any other object included in the same *dlopen()* invocation, and any objects
 8525 that were loaded in any *dlopen()* invocation and which specified the RTLD_GLOBAL flag. To
 8526 determine the scope of visibility for the symbols loaded with a *dlopen()* invocation, the *mode*
 8527 parameter should be a bitwise-inclusive OR with one of the following values:

8528	RTLD_GLOBAL	The object's symbols shall be made available for the relocation processing
8529		of any other object. In addition, symbol lookup using <i>dlopen(0, mode)</i> and
8530		an associated <i>dlsym()</i> allows objects loaded with this <i>mode</i> to be searched.
8531	RTLD_LOCAL	The object's symbols shall not be made available for the relocation
8532		processing of any other object.

8533 If neither RTLD_GLOBAL nor RTLD_LOCAL are specified, then an implementation-defined
 8534 default behavior shall be applied.

8535 If a *file* is specified in multiple *dlopen()* invocations, *mode* is interpreted at each invocation. Note,
 8536 however, that once RTLD_NOW has been specified all relocations shall have been completed
 8537 rendering further RTLD_NOW operations redundant and any further RTLD_LAZY operations
 8538 irrelevant. Similarly, note that once RTLD_GLOBAL has been specified the object shall maintain
 8539 the RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL,
 8540 as long as the object remains in the address space (see *dlclose()*).

8541 Symbols introduced into a program through calls to *dlopen()* may be used in relocation
 8542 activities. Symbols so introduced may duplicate symbols already defined by the program or
 8543 previous *dlopen()* operations. To resolve the ambiguities such a situation might present, the
 8544 resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two
 8545 such resolution orders are defined: *load* or *dependency* ordering. Load order establishes an
 8546 ordering among symbol definitions, such that the definition first loaded (including definitions
 8547 from the image file and any dependent objects loaded with it) has priority over objects added
 8548 later (via *dlopen()*). Load ordering is used in relocation processing. Dependency ordering uses a
 8549 breadth-first order starting with a given object, then all of its dependencies, then any dependents
 8550 of those, iterating until all dependencies are satisfied. With the exception of the global symbol
 8551 object obtained via a *dlopen()* operation on a *file* of 0, dependency ordering is used by the
 8552 *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol object.

8553 When an object is first made accessible via *dlopen()* it and its dependent objects are added in
 8554 dependency order. Once all the objects are added, relocations are performed using load order.
 8555 Note that if an object or its dependencies had been previously loaded, the load and dependency
 8556 orders may yield different resolutions.

8557 The symbols introduced by *dlopen()* operations and available through *dlsym()* are at a minimum
 8558 those which are exported as symbols of global scope by the object. Typically such symbols shall
 8559 be those that were specified in (for example) C source code as having *extern* linkage. The precise
 8560 manner in which an implementation constructs the set of exported symbols for a *dlopen()* object
 8561 is specified by that implementation.

8562 **RETURN VALUE**

8563 If *file* cannot be found, cannot be opened for reading, is not of an appropriate object format for
 8564 processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its
 8565 symbolic references, *dlopen()* shall return NULL. More detailed diagnostic information shall be
 8566 available through *dLError()*.

8567 ERRORS

8568 No errors are defined.

8569 EXAMPLES

8570 None.

8571 APPLICATION USAGE

8572 None.

8573 RATIONALE

8574 None.

8575 FUTURE DIRECTIONS

8576 None.

8577 SEE ALSO

8578 *dlclose()*, *dlderror()*, *dlsym()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**dlfcn.h**>

8579 CHANGE HISTORY

8580 First released in Issue 5.

8581 **NAME**8582 dlsym — obtain the address of a symbol from a *dlopen()* object8583 **SYNOPSIS**8584 **XSI** #include <dlfcn.h>8585 void *dlsym(void *restrict *handle*, const char *restrict *name*);

8586

8587 **DESCRIPTION**

8588 The *dlsym()* function shall obtain the address of a symbol defined within an object made
 8589 accessible through a *dlopen()* call. The *handle* argument is the value returned from a call to
 8590 *dlopen()* (and which has not since been released via a call to *dlclose()*), and *name* is the symbol's
 8591 name as a character string.

8592 The *dlsym()* function shall search for the named symbol in all objects loaded automatically as a
 8593 result of loading the object referenced by *handle* (see *dlopen()*). Load ordering is used in *dlsym()*
 8594 operations upon the global symbol object. The symbol resolution algorithm used shall be
 8595 dependency order as described in *dlopen()*.

8596 The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

8597 **RETURN VALUE**

8598 If *handle* does not refer to a valid object opened by *dlopen()*, or if the named symbol cannot be
 8599 found within any of the objects associated with *handle*, *dlsym()* shall return NULL. More
 8600 detailed diagnostic information shall be available through *dlerror()*.

8601 **ERRORS**

8602 No errors are defined.

8603 **EXAMPLES**

8604 The following example shows how *dlopen()* and *dlsym()* can be used to access either function or
 8605 data objects. For simplicity, error checking has been omitted.

```
8606       void       *handle;
8607       int        *iptr, (*fptr)(int);

8608       /* open the needed object */
8609       handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);

8610       /* find the address of function and data objects */
8611       *(void **)(&fptr) = dlsym(handle, "my_function");
8612       iptr = (int *)dlsym(handle, "my_object");

8613       /* invoke function, passing value of integer as a parameter */
8614       (*fptr)(*iptr);
```

8615 **APPLICATION USAGE**

8616 Special purpose values for *handle* are reserved for future use. These values and their meanings
 8617 are:

8618 RTLD_DEFAULT The symbol lookup happens in the normal global scope; that is, a search for a
 8619 symbol using this handle would find the same definition as a direct use of this
 8620 symbol in the program code.

8621 RTLD_NEXT Specifies the next object after this one that defines *name*. *This one* refers to the
 8622 object containing the invocation of *dlsym()*. The *next* object is the one found
 8623 upon the application of a load order symbol resolution algorithm (see
 8624 *dlopen()*). The next object is either one of global scope (because it was
 8625 introduced as part of the original process image or because it was added with

8626 a *dlopen()* operation including the RTLD_GLOBAL flag), or is an object that
 8627 was included in the same *dlopen()* operation that loaded this one.

8628 The RTLD_NEXT flag is useful to navigate an intentionally created hierarchy
 8629 of multiply-defined symbols created through *interposition*. For example, if a
 8630 program wished to create an implementation of *malloc()* that embedded some
 8631 statistics gathering about memory allocations, such an implementation could
 8632 use the real *malloc()* definition to perform the memory allocation—and itself
 8633 only embed the necessary logic to implement the statistics gathering function.

8634 RATIONALE

8635 The ISO C standard does not require that pointers to functions can be cast back and forth to
 8636 pointers to data. Indeed, the ISO C standard does not require that an object of type **void *** can
 8637 hold a pointer to a function. Implementations supporting the XSI extension, however, do require
 8638 that an object of type **void *** can hold a pointer to a function. The result of converting a pointer to
 8639 a function into a pointer to another data type (except **void ***) is still undefined, however. Note
 8640 that compilers conforming to the ISO C standard are required to generate a warning if a
 8641 conversion from a **void *** pointer to a function pointer is attempted as in:

```
8642 fptr = (int (*)(int))dlsym(handle, "my_function");
```

8643 Due to the problem noted here, a future version may either add a new function to return
 8644 function pointers, or the current interface may be deprecated in favor of two new functions: one
 8645 that returns data pointers and the other that returns function pointers.

8646 FUTURE DIRECTIONS

8647 None.

8648 SEE ALSO

8649 *dlclose()*, *dlderror()*, *dlopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <dlfcn.h>

8650 CHANGE HISTORY

8651 First released in Issue 5.

8652 Issue 6

8653 The **restrict** keyword is added to the *dlsym()* prototype for alignment with the
 8654 ISO / IEC 9899:1999 standard.

8655 The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

8656 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 14 is applied, correcting an example, and
 8657 adding text to the RATIONALE describing issues related to conversion of pointers to functions
 8658 and back again.

NAME

drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS

```

#include <stdlib.h>

double drand48(void);
double erand48(unsigned short xsubi[3]);
long jrand48(unsigned short xsubi[3]);
void lcong48(unsigned short param[7]);
long lrand48(void);
long mrand48(void);
long nrand48(unsigned short xsubi[3]);
unsigned short *seed48(unsigned short seed16v[3]);
void srand48(long seedval);

```

DESCRIPTION

This family of functions shall generate pseudo-random numbers using a linear congruential algorithm and 48-bit integer arithmetic.

The *drand48()* and *erand48()* functions shall return non-negative, double-precision, floating-point values, uniformly distributed over the interval [0.0,1.0).

The *lrand48()* and *nrand48()* functions shall return non-negative, long integers, uniformly distributed over the interval $[0, 2^{31})$.

The *mrnd48()* and *jrnd48()* functions shall return signed long integers uniformly distributed over the interval $[-2^{31}, 2^{31})$.

The *srand48()*, *seed48()*, and *lcong48()* functions are initialization entry points, one of which should be invoked before either *drand48()*, *lrand48()*, or *mrnd48()* is called. (Although it is not recommended practice, constant default initializer values shall be supplied automatically if *drand48()*, *lrand48()*, or *mrnd48()* is called without a prior call to an initialization entry point.) The *erand48()*, *nrand48()*, and *jrnd48()* functions do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, X_i , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless *lcong48()* is invoked, the multiplier value a and the addend value c are given by:

$$a = 5\text{DEECE66D}_{16} = 273673163155_8$$

$$c = \text{B}_{16} = 13_8$$

The value returned by any of the *drand48()*, *erand48()*, *jrnd48()*, *lrand48()*, *mrnd48()*, or *nrand48()* functions is computed by first generating the next 48-bit X_i in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of X_i and transformed into the returned value.

The *drand48()*, *lrand48()*, and *mrnd48()* functions store the last 48-bit X_i generated in an internal buffer; that is why the application shall ensure that these are initialized prior to being invoked. The *erand48()*, *nrand48()*, and *jrnd48()* functions require the calling program to provide storage for the successive X_i values in the array specified as an argument when the

8704 functions are invoked. That is why these routines do not have to be initialized; the calling
 8705 program merely has to place the desired initial value of X_i into the array and pass it as an
 8706 argument. By using different arguments, *erand48()*, *nrand48()*, and *jrand48()* allow separate
 8707 modules of a large program to generate several *independent* streams of pseudo-random numbers;
 8708 that is, the sequence of numbers in each stream shall *not* depend upon how many times the
 8709 routines are called to generate numbers for the other streams.

8710 The initializer function *srand48()* sets the high-order 32 bits of X_i to the low-order 32 bits
 8711 contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

8712 The initializer function *seed48()* sets the value of X_i to the 48-bit value specified in the argument
 8713 array. The low-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[0]. The mid-order 16
 8714 bits of X_i are set to the low-order 16 bits of *seed16v*[1]. The high-order 16 bits of X_i are set to the
 8715 low-order 16 bits of *seed16v*[2]. In addition, the previous value of X_i is copied into a 48-bit
 8716 internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by
 8717 *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is
 8718 to be restarted from a given point at some future time—use the pointer to get at and store the
 8719 last X_i value, and then use this value to reinitialize via *seed48()* when the program is restarted.

8720 The initializer function *lcong48()* allows the user to specify the initial X_i , the multiplier value a ,
 8721 and the addend value c . Argument array elements *param*[0-2] specify X_i , *param*[3-5] specify the
 8722 multiplier a , and *param*[6] specifies the 16-bit addend c . After *lcong48()* is called, a subsequent
 8723 call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values, a and
 8724 c , specified above.

8725 The *drand48()*, *lrand48()*, and *mrnd48()* functions need not be reentrant. A function that is not
 8726 required to be reentrant is not required to be thread-safe.

8727 RETURN VALUE

8728 As described in the DESCRIPTION above.

8729 ERRORS

8730 No errors are defined.

8731 EXAMPLES

8732 None.

8733 APPLICATION USAGE

8734 None.

8735 RATIONALE

8736 None.

8737 FUTURE DIRECTIONS

8738 None.

8739 SEE ALSO

8740 *rand()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

8741 CHANGE HISTORY

8742 First released in Issue 1. Derived from Issue 1 of the SVID.

8743 Issue 5

8744 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8745 Issue 6

8746 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8747 **NAME**

8748 dup, dup2 — duplicate an open file descriptor

8749 **SYNOPSIS**

8750 #include <unistd.h>

8751 int dup(int *fil-des*);8752 int dup2(int *fil-des*, int *fil-des2*);8753 **DESCRIPTION**8754 The *dup()* and *dup2()* functions provide an alternative interface to the service provided by *fcntl()* using the *F_DUPFD* command. The call:8756 *fid* = *dup(fil-des)*;

8757 shall be equivalent to:

8758 *fid* = *fcntl(fil-des, F_DUPFD, 0)*;

8759 The call:

8760 *fid* = *dup2(fil-des, fil-des2)*;

8761 shall be equivalent to:

8762 *close(fil-des2)*;8763 *fid* = *fcntl(fil-des, F_DUPFD, fil-des2)*;

8764 except for the following:

- 8765 • If *fil-des2* is less than 0 or greater than or equal to {*OPEN_MAX*}, *dup2()* shall return *-1* with
- 8766 *errno* set to [*EBADF*].
- 8767 • If *fil-des* is a valid file descriptor and is equal to *fil-des2*, *dup2()* shall return *fil-des2* without
- 8768 closing it.
- 8769 • If *fil-des* is not a valid file descriptor, *dup2()* shall return *-1* and shall not close *fil-des2*.
- 8770 • The value returned shall be equal to the value of *fil-des2* upon successful completion, or *-1*
- 8771 upon failure.

8772 **RETURN VALUE**8773 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned;
8774 otherwise, *-1* shall be returned and *errno* set to indicate the error.8775 **ERRORS**8776 The *dup()* function shall fail if:8777 [*EBADF*] The *fil-des* argument is not a valid open file descriptor.8778 [*EMFILE*] The number of file descriptors in use by this process would exceed
8779 {*OPEN_MAX*}.8780 The *dup2()* function shall fail if:8781 [*EBADF*] The *fil-des* argument is not a valid open file descriptor or the argument *fil-des2* is
8782 negative or greater than or equal to {*OPEN_MAX*}.8783 [*EINTR*] The *dup2()* function was interrupted by a signal.

8784 **EXAMPLES**8785 **Redirecting Standard Output to a File**

8786 The following example closes standard output for the current processes, re-assigns standard
8787 output to go to the file referenced by *pdf*, and closes the original file descriptor to clean up.

```
8788 #include <unistd.h>
8789 ...
8790 int pdf;
8791 ...
8792 close(1);
8793 dup(pdf);
8794 close(pdf);
8795 ...
```

8796 **Redirecting Error Messages**

8797 The following example redirects messages from *stderr* to *stdout*.

```
8798 #include <unistd.h>
8799 ...
8800 dup2(1, 2);
8801 ...
```

8802 **APPLICATION USAGE**

8803 None.

8804 **RATIONALE**

8805 The *dup()* and *dup2()* functions are redundant. Their services are also provided by the *fcntl()*
8806 function. They have been included in this volume of IEEE Std 1003.1-2001 primarily for historical
8807 reasons, since many existing applications use them.

8808 While the brief code segment shown is very similar in behavior to *dup2()*, a conforming
8809 implementation based on other functions defined in this volume of IEEE Std 1003.1-2001 is
8810 significantly more complex. Least obvious is the possible effect of a signal-catching function that
8811 could be invoked between steps and allocate or deallocate file descriptors. This could be avoided
8812 by blocking signals.

8813 The *dup2()* function is not marked obsolescent because it presents a type-safe version of
8814 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

8815 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

8816 In the description of [EBADF], the case of *fildes* being out of range is covered by the given case of
8817 *fildes* not being valid. The descriptions for *fildes* and *fildes2* are different because the only kind of
8818 invalidity that is relevant for *fildes2* is whether it is out of range; that is, it does not matter
8819 whether *fildes2* refers to an open file when the *dup2()* call is made.

8820 **FUTURE DIRECTIONS**

8821 None.

8822 **SEE ALSO**

8823 *close()*, *fcntl()*, *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

8824 **CHANGE HISTORY**

8825 First released in Issue 1. Derived from Issue 1 of the SVID.

8826 **NAME**8827 ecvt, fcvt, gcvt — convert a floating-point number to a string (**LEGACY**)8828 **SYNOPSIS**

8829 xSI #include <stdlib.h>

```

8830     char *ecvt(double value, int ndigit, int *restrict decpt,
8831                int *restrict sign);
8832     char *fcvt(double value, int ndigit, int *restrict decpt,
8833                int *restrict sign);
8834     char *gcvt(double value, int ndigit, char *buf);
8835 
```

8836 **DESCRIPTION**8837 The *ecvt()*, *fcvt()*, and *gcvt()* functions shall convert floating-point numbers to null-terminated strings.

8839 The *ecvt()* function shall convert *value* to a null-terminated string of *ndigit* digits (where *ndigit* is reduced to an unspecified limit determined by the precision of a **double**) and return a pointer to the string. The high-order digit shall be non-zero, unless the value is 0. The low-order digit shall be rounded in an implementation-defined manner. The position of the radix character relative to the beginning of the string shall be stored in the integer pointed to by *decpt* (negative means to the left of the returned digits). If *value* is zero, it is unspecified whether the integer pointed to by *decpt* would be 0 or 1. The radix character shall not be included in the returned string. If the sign of the result is negative, the integer pointed to by *sign* shall be non-zero; otherwise, it shall be 0.

8847 If the converted value is out of range or is not representable, the contents of the returned string are unspecified.

8849 The *fcvt()* function shall be equivalent to *ecvt()*, except that *ndigit* specifies the number of digits desired after the radix character. The total number of digits in the result string is restricted to an unspecified limit as determined by the precision of a **double**.

8852 The *gcvt()* function shall convert *value* to a null-terminated string (similar to that of the %g conversion specification format of *printf()*) in the array pointed to by *buf* and shall return *buf*. It shall produce *ndigit* significant digits (limited to an unspecified value determined by the precision of a **double**) in the %F conversion specification format of *printf()* if possible, or the %e conversion specification format of *printf()* (scientific notation) otherwise. A minus sign shall be included in the returned string if *value* is less than 0. A radix character shall be included in the returned string if *value* is not a whole number. Trailing zeros shall be suppressed where *value* is not a whole number. The radix character is determined by the current locale. If *setlocale()* has not been called successfully, the default locale, POSIX, is used. The default locale specifies a period ('.') as the radix character. The *LC_NUMERIC* category determines the value of the radix character within the current locale.

8863 These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

8865 **RETURN VALUE**8866 The *ecvt()* and *fcvt()* functions shall return a pointer to a null-terminated string of digits.8867 The *gcvt()* function shall return *buf*.

8868 The return values from *ecvt()* and *fcvt()* may point to static data which may be overwritten by subsequent calls to these functions.

8870 ERRORS

8871 No errors are defined.

8872 EXAMPLES

8873 None.

8874 APPLICATION USAGE

8875 The *sprintf()* function is preferred over this function.

8876 RATIONALE

8877 None.

8878 FUTURE DIRECTIONS

8879 These functions may be withdrawn in a future version.

8880 SEE ALSO

8881 *printf()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

8882 CHANGE HISTORY

8883 First released in Issue 4, Version 2.

8884 Issue 5

8885 Moved from X/OPEN UNIX extension to BASE.

8886 Normative text previously in the APPLICATION USAGE section is moved to the
8887 DESCRIPTION.

8888 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8889 Issue 6

8890 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8891 This function is marked LEGACY.

8892 The **restrict** keyword is added to the *ecvt()* and *fcvt()* prototypes for alignment with the
8893 ISO/IEC 9899:1999 standard.

8894 The DESCRIPTION is updated to explicitly use “conversion specification” to describe %g, %f,
8895 and %e.

8896 **NAME**8897 encrypt — encoding function (**CRYPT**)8898 **SYNOPSIS**

8899 xSI #include <unistd.h>

8900 void encrypt(char block[64], int edflag);

8901

8902 **DESCRIPTION**

8903 The *encrypt()* function shall provide access to an implementation-defined encoding algorithm.
8904 The key generated by *setkey()* is used to encrypt the string *block* with *encrypt()*.

8905 The *block* argument to *encrypt()* shall be an array of length 64 bytes containing only the bytes
8906 with values of 0 and 1. The array is modified in place to a similar array using the key set by
8907 *setkey()*. If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see
8908 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to
8909 [ENOSYS].

8910 The *encrypt()* function shall not change the setting of *errno* if successful. An application wishing
8911 to check for error situations should set *errno* to 0 before calling *encrypt()*. If *errno* is non-zero on
8912 return, an error has occurred.

8913 The *encrypt()* function need not be reentrant. A function that is not required to be reentrant is
8914 not required to be thread-safe.

8915 **RETURN VALUE**8916 The *encrypt()* function shall not return a value.8917 **ERRORS**8918 The *encrypt()* function shall fail if:

8919 [ENOSYS] The functionality is not supported on this implementation.

8920 **EXAMPLES**

8921 None.

8922 **APPLICATION USAGE**8923 Historical implementations of the *encrypt()* function used a rather primitive encoding algorithm.

8924 In some environments, decoding might not be implemented. This is related to some Government
8925 restrictions on encryption and decryption routines. Historical practice has been to ship a
8926 different version of the encryption library without the decryption feature in the routines
8927 supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

8928 **RATIONALE**

8929 None.

8930 **FUTURE DIRECTIONS**

8931 None.

8932 **SEE ALSO**8933 *crypt()*, *setkey()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>8934 **CHANGE HISTORY**

8935 First released in Issue 1. Derived from Issue 1 of the SVID.

8936 **Issue 5**

8937 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

8938 **Issue 6**

8939 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8940 **NAME**

8941 endgrent, getgrent, setgrent — group database entry functions

8942 **SYNOPSIS**8943 xSI `#include <grp.h>`

```
8944 void endgrent(void);
8945 struct group *getgrent(void);
8946 void setgrent(void);
8947
```

8948 **DESCRIPTION**

8949 The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an
 8950 entry in the group database. When first called, *getgrent()* shall return a pointer to a **group**
 8951 structure containing the first entry in the group database. Thereafter, it shall return a pointer to a
 8952 **group** structure containing the next group structure in the group database, so successive calls
 8953 may be used to search the entire database.

8954 An implementation that provides extended security controls may impose further
 8955 implementation-defined restrictions on accessing the group database. In particular, the system
 8956 may deny the existence of some or all of the group database entries associated with groups other
 8957 than those groups associated with the caller and may omit users other than the caller from the
 8958 list of members of groups in database entries that are returned.

8959 The *setgrent()* function shall rewind the group database to allow repeated searches.

8960 The *endgrent()* function may be called to close the group database when processing is complete.

8961 These functions need not be reentrant. A function that is not required to be reentrant is not
 8962 required to be thread-safe.

8963 **RETURN VALUE**

8964 When first called, *getgrent()* shall return a pointer to the first group structure in the group
 8965 database. Upon subsequent calls it shall return the next group structure in the group database.
 8966 The *getgrent()* function shall return a null pointer on end-of-file or an error and *errno* may be set
 8967 to indicate the error.

8968 The return value may point to a static area which is overwritten by a subsequent call to
 8969 *getgrgid()*, *getgrnam()*, or *getgrent()*.

8970 **ERRORS**

8971 The *getgrent()* function may fail if:

8972	[EINTR]	A signal was caught during the operation.
8973	[EIO]	An I/O error has occurred.
8974	[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
8975	[ENFILE]	The maximum allowable number of files is currently open in the system.

8976 EXAMPLES

8977 None.

8978 APPLICATION USAGE

8979 These functions are provided due to their historical usage. Applications should avoid
8980 dependencies on fields in the group database, whether the database is a single file, or where in
8981 the file system name space the database resides. Applications should use *getgrnam()* and
8982 *getgrgid()* whenever possible because it avoids these dependencies.

8983 RATIONALE

8984 None.

8985 FUTURE DIRECTIONS

8986 None.

8987 SEE ALSO

8988 *getgrgid()*, *getgrnam()*, *getlogin()*, *getpwent()*, the Base Definitions volume of
8989 IEEE Std 1003.1-2001, <grp.h>

8990 CHANGE HISTORY

8991 First released in Issue 4, Version 2.

8992 Issue 5

8993 Moved from X/OPEN UNIX extension to BASE.

8994 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
8995 VALUE section.

8996 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8997 Issue 6

8998 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8999 **NAME**

9000 endhostent, gethostent, sethostent — network host database functions

9001 **SYNOPSIS**

9002 #include <netdb.h>

9003 void endhostent(void);

9004 struct hostent *gethostent(void);

9005 void sethostent(int stayopen);

9006 **DESCRIPTION**

9007 These functions shall retrieve information about hosts. This information is considered to be
9008 stored in a database that can be accessed sequentially or randomly. The implementation of this
9009 database is unspecified.

9010 **Note:** In many cases this database is implemented by the Domain Name System, as documented in
9011 RFC 1034, RFC 1035, and RFC 1886.

9012 The *sethostent()* function shall open a connection to the database and set the next entry for
9013 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection
9014 shall not be closed by a call to *gethostent()*, *gethostbyname()*, or *gethostbyaddr()*, and the
9015 implementation may maintain an open file descriptor.

9016 The *gethostent()* function shall read the next entry in the database, opening and closing a
9017 connection to the database as necessary.

9018 Entries shall be returned in **hostent** structures. Refer to *gethostbyaddr()* for a definition of the
9019 **hostent** structure.

9020 The *endhostent()* function shall close the connection to the database, releasing any open file
9021 descriptor.

9022 These functions need not be reentrant. A function that is not required to be reentrant is not
9023 required to be thread-safe.

9024 **RETURN VALUE**

9025 Upon successful completion, the *gethostent()* function shall return a pointer to a **hostent**
9026 structure if the requested entry was found, and a null pointer if the end of the database was
9027 reached or the requested entry was not found.

9028 **ERRORS**9029 No errors are defined for *endhostent()*, *gethostent()*, and *sethostent()*.9030 **EXAMPLES**

9031 None.

9032 **APPLICATION USAGE**

9033 The *gethostent()* function may return pointers to static data, which may be overwritten by
9034 subsequent calls to any of these functions.

9035 **RATIONALE**

9036 None.

9037 **FUTURE DIRECTIONS**

9038 None.

9039 **SEE ALSO**9040 *endservent()*, *gethostbyaddr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <netdb.h>

9041 **CHANGE HISTORY**

9042 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9043 **NAME**

9044 endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

9045 **SYNOPSIS**

9046 #include <netdb.h>

9047 void endnetent(void);

9048 struct netent *getnetbyaddr(uint32_t net, int type);

9049 struct netent *getnetbyname(const char *name);

9050 struct netent *getnetent(void);

9051 void setnetent(int stayopen);

9052 **DESCRIPTION**

9053 These functions shall retrieve information about networks. This information is considered to be
 9054 stored in a database that can be accessed sequentially or randomly. The implementation of this
 9055 database is unspecified.

9056 The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-
 9057 zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either
 9058 directly, or indirectly through one of the other *getnet**(*)* functions), and the implementation may
 9059 maintain an open file descriptor to the database.

9060 The *getnetent()* function shall read the next entry of the database, opening and closing a
 9061 connection to the database as necessary.

9062 The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry
 9063 for which the address family specified by *type* matches the *n_addrtype* member and the network
 9064 number *net* matches the *n_net* member, opening and closing a connection to the database as
 9065 necessary. The *net* argument shall be the network number in host byte order.

9066 The *getnetbyname()* function shall search the database from the beginning and find the first entry
 9067 for which the network name specified by *name* matches the *n_name* member, opening and
 9068 closing a connection to the database as necessary.

9069 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions shall each return a pointer to a
 9070 **netent** structure, the members of which shall contain the fields of an entry in the network
 9071 database.

9072 The *endnetent()* function shall close the database, releasing any open file descriptor.

9073 These functions need not be reentrant. A function that is not required to be reentrant is not
 9074 required to be thread-safe.

9075 **RETURN VALUE**

9076 Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* shall return a
 9077 pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the
 9078 database was reached or the requested entry was not found. Otherwise, a null pointer shall be
 9079 returned.

9080 **ERRORS**

9081 No errors are defined.

9082 **EXAMPLES**

9083 None.

9084 **APPLICATION USAGE**

9085 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions may return pointers to static data,
9086 which may be overwritten by subsequent calls to any of these functions.

9087 **RATIONALE**

9088 None.

9089 **FUTURE DIRECTIONS**

9090 None.

9091 **SEE ALSO**9092 The Base Definitions volume of IEEE Std 1003.1-2001, <**netdb.h**>9093 **CHANGE HISTORY**

9094 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9095 **NAME**

9096 endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol
 9097 database functions

9098 **SYNOPSIS**

```
9099     #include <netdb.h>

9100     void endprotoent(void);
9101     struct protoent *getprotobyname(const char *name);
9102     struct protoent *getprotobynumber(int proto);
9103     struct protoent *getprotoent(void);
9104     void setprotoent(int stayopen);
```

9105 **DESCRIPTION**

9106 These functions shall retrieve information about protocols. This information is considered to be
 9107 stored in a database that can be accessed sequentially or randomly. The implementation of this
 9108 database is unspecified.

9109 The *setprotoent()* function shall open a connection to the database, and set the next entry to the
 9110 first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database
 9111 shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the
 9112 other *getproto**(*)* functions), and the implementation may maintain an open file descriptor for
 9113 the database.

9114 The *getprotobyname()* function shall search the database from the beginning and find the first
 9115 entry for which the protocol name specified by *name* matches the *p_name* member, opening and
 9116 closing a connection to the database as necessary.

9117 The *getprotobynumber()* function shall search the database from the beginning and find the first
 9118 entry for which the protocol number specified by *proto* matches the *p_proto* member, opening
 9119 and closing a connection to the database as necessary.

9120 The *getprotoent()* function shall read the next entry of the database, opening and closing a
 9121 connection to the database as necessary.

9122 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions shall each return a pointer
 9123 to a **protoent** structure, the members of which shall contain the fields of an entry in the network
 9124 protocol database.

9125 The *endprotoent()* function shall close the connection to the database, releasing any open file
 9126 descriptor.

9127 These functions need not be reentrant. A function that is not required to be reentrant is not
 9128 required to be thread-safe.

9129 **RETURN VALUE**

9130 Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a
 9131 pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of
 9132 the database was reached or the requested entry was not found. Otherwise, a null pointer is
 9133 returned.

9134 **ERRORS**

9135 No errors are defined.

9136 **EXAMPLES**

9137 None.

9138 **APPLICATION USAGE**

9139 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions may return pointers to
9140 static data, which may be overwritten by subsequent calls to any of these functions.

9141 **RATIONALE**

9142 None.

9143 **FUTURE DIRECTIONS**

9144 None.

9145 **SEE ALSO**9146 The Base Definitions volume of IEEE Std 1003.1-2001, <**netdb.h**>9147 **CHANGE HISTORY**

9148 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9149 **NAME**

9150 endpwent, getpwent, setpwent — user database functions

9151 **SYNOPSIS**9152 xSI `#include <pwd.h>`9153 `void endpwent(void);`9154 `struct passwd *getpwent(void);`9155 `void setpwent(void);`

9156

9157 **DESCRIPTION**

9158 These functions shall retrieve information about users.

9159 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of
 9160 an entry in the user database. Each entry in the user database contains a **passwd** structure. When
 9161 first called, *getpwent()* shall return a pointer to a **passwd** structure containing the first entry in
 9162 the user database. Thereafter, it shall return a pointer to a **passwd** structure containing the next
 9163 entry in the user database. Successive calls can be used to search the entire user database.

9164 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.

9165 An implementation that provides extended security controls may impose further
 9166 implementation-defined restrictions on accessing the user database. In particular, the system
 9167 may deny the existence of some or all of the user database entries associated with users other
 9168 than the caller.

9169 The *setpwent()* function effectively rewinds the user database to allow repeated searches.9170 The *endpwent()* function may be called to close the user database when processing is complete.

9171 These functions need not be reentrant. A function that is not required to be reentrant is not
 9172 required to be thread-safe.

9173 **RETURN VALUE**9174 The *getpwent()* function shall return a null pointer on end-of-file or error.9175 **ERRORS**9176 The *getpwent()*, *setpwent()*, and *endpwent()* functions may fail if:

9177 [EIO] An I/O error has occurred.

9178 In addition, *getpwent()* and *setpwent()* may fail if:

9179 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

9180 [ENFILE] The maximum allowable number of files is currently open in the system.

9181 The return value may point to a static area which is overwritten by a subsequent call to
 9182 *getpwuid()*, *getpwnam()*, or *getpwent()*.

9183 **EXAMPLES**9184 **Searching the User Database**

9185 The following example uses the *getpwent()* function to get successive entries in the user
9186 database, returning a pointer to a **passwd** structure that contains information about each user.
9187 The call to *endpwent()* closes the user database and cleans up.

```
9188 #include <pwd.h>
9189 ...
9190 struct passwd *p;
9191 ...
9192 while ((p = getpwent ()) != NULL) {
9193     ...
9194 }
9195 endpwent();
9196 ...
```

9197 **APPLICATION USAGE**

9198 These functions are provided due to their historical usage. Applications should avoid
9199 dependencies on fields in the password database, whether the database is a single file, or where
9200 in the file system name space the database resides. Applications should use *getpwuid()*
9201 whenever possible because it avoids these dependencies.

9202 **RATIONALE**

9203 None.

9204 **FUTURE DIRECTIONS**

9205 None.

9206 **SEE ALSO**

9207 *endgrent()*, *getlogin()*, *getpwnam()*, *getpwuid()*, the Base Definitions volume of
9208 IEEE Std 1003.1-2001, **<pwd.h>**

9209 **CHANGE HISTORY**

9210 First released in Issue 4, Version 2.

9211 **Issue 5**

9212 Moved from X/OPEN UNIX extension to BASE.

9213 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
9214 VALUE section.

9215 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9216 **Issue 6**

9217 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9218 **NAME**

9219 endservent, getservbyname, getservbyport, getservent, setservent — network services database
9220 functions

9221 **SYNOPSIS**

```
9222 #include <netdb.h>

9223 void endservent(void);
9224 struct servent *getservbyname(const char *name, const char *proto);
9225 struct servent *getservbyport(int port, const char *proto);
9226 struct servent *getservent(void);
9227 void setservent(int stayopen);
```

9228 **DESCRIPTION**

9229 These functions shall retrieve information about network services. This information is
9230 considered to be stored in a database that can be accessed sequentially or randomly. The
9231 implementation of this database is unspecified.

9232 The *setservent()* function shall open a connection to the database, and set the next entry to the
9233 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each
9234 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv*()*
9235 functions), and the implementation may maintain an open file descriptor for the database.

9236 The *getservent()* function shall read the next entry of the database, opening and closing a
9237 connection to the database as necessary.

9238 The *getservbyname()* function shall search the database from the beginning and find the first
9239 entry for which the service name specified by *name* matches the *s_name* member and the protocol
9240 name specified by *proto* matches the *s_proto* member, opening and closing a connection to the
9241 database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be
9242 matched.

9243 The *getservbyport()* function shall search the database from the beginning and find the first entry
9244 for which the port specified by *port* matches the *s_port* member and the protocol name specified
9245 by *proto* matches the *s_proto* member, opening and closing a connection to the database as
9246 necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched. The *port*
9247 argument shall be in network byte order.

9248 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a
9249 **servent** structure, the members of which shall contain the fields of an entry in the network
9250 services database.

9251 The *endservent()* function shall close the database, releasing any open file descriptor.

9252 These functions need not be reentrant. A function that is not required to be reentrant is not
9253 required to be thread-safe.

9254 **RETURN VALUE**

9255 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to
9256 a **servent** structure if the requested entry was found, and a null pointer if the end of the database
9257 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

9258 **ERRORS**

9259 No errors are defined.

9260 **EXAMPLES**

9261 None.

9262 **APPLICATION USAGE**9263 The *port* argument of *getservbyport()* need not be compatible with the port values of all address
9264 families.9265 The *getservbyname()*, *getservbyport()*, and *getservent()* functions may return pointers to static
9266 data, which may be overwritten by subsequent calls to any of these functions.9267 **RATIONALE**

9268 None.

9269 **FUTURE DIRECTIONS**

9270 None.

9271 **SEE ALSO**9272 *endhostent()*, *endprotoent()*, *htonl()*, *inet_addr()*, the Base Definitions volume of
9273 IEEE Std 1003.1-2001, <**netdb.h**>9274 **CHANGE HISTORY**

9275 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9276 **NAME**

9277 endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database
 9278 functions

9279 **SYNOPSIS**

```
9280 xSI #include <utmpx.h>

9281 void endutxent(void);
9282 struct utmpx *getutxent(void);
9283 struct utmpx *getutxid(const struct utmpx *id);
9284 struct utmpx *getutxline(const struct utmpx *line);
9285 struct utmpx *pututxline(const struct utmpx *utmpx);
9286 void setutxent(void);
9287
```

9288 **DESCRIPTION**

9289 These functions shall provide access to the user accounting database.

9290 The *getutxent()* function shall read the next entry from the user accounting database. If the
 9291 database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

9292 The *getutxid()* function shall search forward from the current point in the database. If the
 9293 *ut_type* value of the **utmpx** structure pointed to by *id* is *BOOT_TIME*, *OLD_TIME*, or
 9294 *NEW_TIME*, then it shall stop when it finds an entry with a matching *ut_type* value. If the
 9295 *ut_type* value is *INIT_PROCESS*, *LOGIN_PROCESS*, *USER_PROCESS*, or *DEAD_PROCESS*,
 9296 then it shall stop when it finds an entry whose type is one of these four and whose *ut_id* member
 9297 matches the *ut_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is
 9298 reached without a match, *getutxid()* shall fail.

9299 The *getutxline()* function shall search forward from the current point in the database until it
 9300 finds an entry of the type *LOGIN_PROCESS* or *USER_PROCESS* which also has a *ut_line* value
 9301 matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached
 9302 without a match, *getutxline()* shall fail.

9303 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to
 9304 search for multiple occurrences, the application shall zero out the static data after each success,
 9305 or *getutxline()* may return a pointer to the same **utmpx** structure.

9306 There is one exception to the rule about clearing the structure before further reads are done. The
 9307 implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the user
 9308 accounting database) shall not modify the static structure returned by *getutxent()*, *getutxid()*, or
 9309 *getutxline()*, if the application has modified this structure and passed the pointer back to
 9310 *pututxline()*.

9311 For all entries that match a request, the *ut_type* member indicates the type of the entry. Other
 9312 members of the entry shall contain meaningful data based on the value of the *ut_type* member as
 9313 follows:

ut_type Member	Other Members with Meaningful Data
EMPTY	No others
BOOT_TIME	<i>ut_tv</i>
OLD_TIME	<i>ut_tv</i>
NEW_TIME	<i>ut_tv</i>
USER_PROCESS	<i>ut_id</i> , <i>ut_user</i> (login name of the user), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
INIT_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>
LOGIN_PROCESS	<i>ut_id</i> , <i>ut_user</i> (implementation-defined name of the login process), <i>ut_pid</i> , <i>ut_tv</i>
DEAD_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>

An implementation that provides extended security controls may impose implementation-defined restrictions on accessing the user accounting database. In particular, the system may deny the existence of some or all of the user accounting database entries associated with users other than the caller.

If the process has appropriate privileges, the *pututxline()* function shall write out the structure into the user accounting database. It shall use *getutxid()* to search for a record that satisfies the request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be made at the end of the user accounting database.

The *endutxent()* function shall close the user accounting database.

The *setutxent()* function shall reset the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

The return value may point to a static area which is overwritten by a subsequent call to *getutxid()* or *getutxline()*.

Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

The *endutxent()* and *setutxent()* functions shall not return a value.

ERRORS

No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()* functions.

The *pututxline()* function may fail if:

[EPERM] The process does not have appropriate privileges.

9353 **EXAMPLES**

9354 None.

9355 **APPLICATION USAGE**9356 The sizes of the arrays in the structure can be found using the *sizeof* operator.9357 **RATIONALE**

9358 None.

9359 **FUTURE DIRECTIONS**

9360 None.

9361 **SEE ALSO**

9362 The Base Definitions volume of IEEE Std 1003.1-2001, <utmpx.h>

9363 **CHANGE HISTORY**

9364 First released in Issue 4, Version 2.

9365 **Issue 5**

9366 Moved from X/OPEN UNIX extension to BASE.

9367 Normative text previously in the APPLICATION USAGE section is moved to the
9368 DESCRIPTION.

9369 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9370 **Issue 6**

9371 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9372 **NAME**9373 `environ` — array of character pointers to the environment strings9374 **SYNOPSIS**9375 `extern char **environ;`9376 **DESCRIPTION**9377 Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables
9378 and *exec*.

9379 **NAME**

9380 erand48 — generate uniformly distributed pseudo-random numbers

9381 **SYNOPSIS**

9382 xSI #include <stdlib.h>

9383 double erand48(unsigned short xsubi[3]);

9384

9385 **DESCRIPTION**9386 Refer to *drand48()*.

9387 **NAME**

9388 erf, erff, erfl — error functions

9389 **SYNOPSIS**

9390 #include <math.h>

9391 double erf(double x);

9392 float erff(float x);

9393 long double erfl(long double x);

9394 **DESCRIPTION**

9395 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 9396 conflict between the requirements described here and the ISO C standard is unintentional. This
 9397 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

9398 These functions shall compute the error function of their argument x , defined as:

$$9399 \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

9400 An application wishing to check for error situations should set *errno* to zero and call
 9401 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 9402 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 9403 zero, an error has occurred.

9404 **RETURN VALUE**

9405 Upon successful completion, these functions shall return the value of the error function.

9406 MX If x is NaN, a NaN shall be returned.

9407 If x is ± 0 , ± 0 shall be returned.

9408 If x is $\pm \text{Inf}$, ± 1 shall be returned.

9409 If x is subnormal, a range error may occur, and $2 * x / \text{sqrt}(\pi)$ should be returned.

9410 **ERRORS**

9411 These functions may fail if:

9412 MX **Range Error** The result underflows.

9413 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 9414 then *errno* shall be set to [ERANGE]. If the integer expression
 9415 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 9416 floating-point exception shall be raised.

9417 **EXAMPLES**

9418 None.

9419 **APPLICATION USAGE**

9420 Underflow occurs when $|x| < \text{DBL\_MIN} * (\text{sqrt}(\pi) / 2)$.

9421 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 9422 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

9423 **RATIONALE**

9424 None.

9425 **FUTURE DIRECTIONS**

9426 None.

9427 **SEE ALSO**

9428 *erfc()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
9429 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

9430 **CHANGE HISTORY**

9431 First released in Issue 1. Derived from Issue 1 of the SVID.

9432 **Issue 5**

9433 The DESCRIPTION is updated to indicate how an application should check for an error. This
9434 text was previously published in the APPLICATION USAGE section.

9435 **Issue 6**9436 The *erf()* function is no longer marked as an extension.9437 The *erfc()* function is split out onto its own reference page. |9438 The *erfff()* and *erfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

9439 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
9440 revised to align with the ISO/IEC 9899:1999 standard.

9441 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
9442 marked.

9443 **NAME**

9444 erfc, erfcf, erfcl — complementary error functions

9445 **SYNOPSIS**

9446 #include <math.h>

9447 double erfc(double x);

9448 float erfcf(float x);

9449 long double erfcl(long double x);

9450 **DESCRIPTION**

9451 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 9452 conflict between the requirements described here and the ISO C standard is unintentional. This
 9453 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

9454 These functions shall compute the complementary error function $1.0 - \text{erf}(x)$.

9455 An application wishing to check for error situations should set *errno* to zero and call
 9456 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 9457 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 9458 zero, an error has occurred.

9459 **RETURN VALUE**9460 Upon successful completion, these functions shall return the value of the complementary error
9461 function.

9462 If the correct value would cause underflow and is not representable, a range error may occur
 9463 **MX** and either 0.0 (if representable), or an implementation-defined value shall be returned.

9464 **MX** If *x* is NaN, a NaN shall be returned.9465 If *x* is ± 0 , +1 shall be returned.9466 If *x* is $-\text{Inf}$, +2 shall be returned.9467 If *x* is $+\text{Inf}$, +0 shall be returned.

9468 If the correct value would cause underflow and is representable, a range error may occur and the
 9469 correct value shall be returned.

9470 **ERRORS**

9471 These functions may fail if:

9472 Range Error The result underflows.

9473 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 9474 then *errno* shall be set to [ERANGE]. If the integer expression
 9475 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 9476 floating-point exception shall be raised.

9477 **EXAMPLES**

9478 None.

9479 **APPLICATION USAGE**

9480 The *erfc()* function is provided because of the extreme loss of relative accuracy if *erf(x)* is called
 9481 for large *x* and the result subtracted from 1.0.

9482 Note for IEEE Std 754-1985 **double**, $26.55 < x$ implies *erfc(x)* has underflowed.

9483 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 9484 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

9485 **RATIONALE**

9486 None.

9487 **FUTURE DIRECTIONS**

9488 None.

9489 **SEE ALSO**

9490 *erf()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
9491 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

9492 **CHANGE HISTORY**

9493 First released in Issue 1. Derived from Issue 1 of the SVID.

9494 **Issue 5**

9495 The DESCRIPTION is updated to indicate how an application should check for an error. This
9496 text was previously published in the APPLICATION USAGE section.

9497 **Issue 6**9498 The *erfc()* function is no longer marked as an extension.9499 These functions are split out from the *erf()* reference page.

9500 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
9501 revised to align with the ISO/IEC 9899:1999 standard.

9502 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
9503 marked.

9504 **NAME**

9505 erff, erfl — error functions

9506 **SYNOPSIS**

9507 #include <math.h>

9508 float erff(float x);

9509 long double erfl(long double x);

9510 **DESCRIPTION**9511 Refer to *erf()*.

9512 **NAME**

9513 errno — error return value

9514 **SYNOPSIS**

9515 #include <errno.h>

9516 **DESCRIPTION**9517 The lvalue *errno* is used by many functions to return error values.

9518 Many functions provide an error number in *errno*, which has type **int** and is defined in
9519 <**errno.h**>. The value of *errno* shall be defined only after a call to a function for which it is
9520 explicitly stated to be set and until it is changed by the next function call or if the application
9521 assigns it a value. The value of *errno* should only be examined when it is indicated to be valid by
9522 a function's return value. Applications shall obtain the definition of *errno* by the inclusion of
9523 <**errno.h**>. No function in this volume of IEEE Std 1003.1-2001 shall set *errno* to 0.

9524 It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a
9525 macro definition is suppressed in order to access an actual object, or a program defines an
9526 identifier with the name *errno*, the behavior is undefined.

9527 The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant
9528 pages.

9529 **RETURN VALUE**

9530 None.

9531 **ERRORS**

9532 None.

9533 **EXAMPLES**

9534 None.

9535 **APPLICATION USAGE**

9536 Previously both POSIX and X/Open documents were more restrictive than the ISO C standard
9537 in that they required *errno* to be defined as an external variable, whereas the ISO C standard
9538 required only that *errno* be defined as a modifiable lvalue with type **int**.

9539 An application that needs to examine the value of *errno* to determine the error should set it to 0
9540 before a function call, then inspect it before a subsequent function call.

9541 **RATIONALE**

9542 None.

9543 **FUTURE DIRECTIONS**

9544 None.

9545 **SEE ALSO**9546 Section 2.3, the Base Definitions volume of IEEE Std 1003.1-2001, <**errno.h**>9547 **CHANGE HISTORY**

9548 First released in Issue 1. Derived from Issue 1 of the SVID.

9549 **Issue 5**

9550 The following sentence is deleted from the DESCRIPTION: “The value of *errno* is 0 at program
9551 start-up, but is never set to 0 by any XSI function”. The DESCRIPTION also no longer states that
9552 conforming implementations may support the declaration:

9553 extern int errno;

9554 **Issue 6**

9555 Obsolescent text regarding defining *errno* as:

9556 `extern int errno`

9557 is removed.

9558 Text regarding no function setting *errno* to zero to indicate an error is changed to no function
9559 shall set *errno* to zero. This is for alignment with the ISO / IEC 9899:1999 standard.

NAME

environ, execl, execv, execl, execve, execlp, execvp — execute a file

SYNOPSIS

```
#include <unistd.h>

extern char **environ;
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execl(const char *path, const char *arg0, ... /*,
(char *)0, char *const envp[] */);
int execve(const char *path, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execvp(const char *file, char *const argv[]);
```

DESCRIPTION

The *exec* family of functions shall replace the current process image with a new process image. The new image shall be constructed from a regular, executable file called the *new process image file*. There shall be no return from a successful *exec*, because the calling process image is overlaid by the new process image.

When a C-language program is executed as a result of this call, it shall be entered as a C-language function call as follows:

```
int main (int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable:

```
extern char **environ;
```

is initialized as a pointer to an array of character pointers to the environment strings. The *argv* and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv* array is not counted in *argc*.

THR Conforming multi-threaded applications shall not use the *environ* variable to access or modify any environment variable while any other thread is concurrently modifying any environment variable. A call to any function dependent on any environment variable shall be considered a use of the *environ* variable to access that environment variable.

The arguments specified by a program with one of the *exec* functions shall be passed on to the new process image in the corresponding *main()* arguments.

The argument *path* points to a pathname that identifies the new process image file.

The argument *file* is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, the *file* argument shall be used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable *PATH* (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables). If this environment variable is not present, the results of the search are implementation-defined.

There are two distinct ways in which the contents of the process image file may cause the execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the ERRORS section). In the cases where the other members of the *exec* family of functions would fail and set *errno* to [ENOEXEC], the *execlp()* and *execvp()* functions shall execute a command interpreter and the environment of the executed command shall be as if the process invoked the *sh* utility using *execl()* as follows:

9605 `execl(<shell path>, arg0, file, arg1, ..., (char *)0);`

9606 where *<shell path>* is an unspecified pathname for the *sh* utility, *file* is the process image file, and
 9607 for *execvp()*, where *arg0*, *arg1*, and so on correspond to the values passed to *execvp()* in *argv*[0],
 9608 *argv*[1], and so on.

9609 The arguments represented by *arg0*,... are pointers to null-terminated character strings. These
 9610 strings shall constitute the argument list available to the new process image. The list is
 9611 terminated by a null pointer. The argument *arg0* should point to a filename that is associated
 9612 with the process being started by one of the *exec* functions.

9613 The argument *argv* is an array of character pointers to null-terminated strings. The application
 9614 shall ensure that the last member of this array is a null pointer. These strings shall constitute the
 9615 argument list available to the new process image. The value in *argv*[0] should point to a filename
 9616 that is associated with the process being started by one of the *exec* functions.

9617 The argument *envp* is an array of character pointers to null-terminated strings. These strings
 9618 shall constitute the environment for the new process image. The *envp* array is terminated by a
 9619 null pointer.

9620 For those forms not containing an *envp* pointer (*execl()*, *execv()*, *execlp()*, and *execvp()*), the
 9621 environment for the new process image shall be taken from the external variable *environ* in the
 9622 calling process.

9623 The number of bytes available for the new process' combined argument and environment lists is
 9624 {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any
 9625 alignment bytes are included in this total.

9626 File descriptors open in the calling process image shall remain open in the new process image,
 9627 except for those whose close-on-exec flag FD_CLOEXEC is set. For those file descriptors that
 9628 remain open, all attributes of the open file description remain unchanged. For any file descriptor
 9629 that is closed for this reason, file locks are removed as a result of the close as described in *close()*.
 9630 Locks that are not removed by closing of file descriptors remain unchanged.

9631 If file descriptors 0, 1, and 2 would otherwise be closed after a successful call to one of the *exec* |
 9632 family of functions, and the new process image file has the set-user-ID or set-group-ID file mode |
 9633 bits set, and the ST_NOSUID bit is not set for the file system containing the new process image |
 9634 file, implementations may open an unspecified file for each of these file descriptors in the new |
 9635 process image. |

9636 Directory streams open in the calling process image shall be closed in the new process image.

9637 The state of the floating-point environment in the new process image shall be set to the default.

9638 XSI The state of conversion descriptors and message catalog descriptors in the new process image is
 9639 undefined. For the new process image, the equivalent of:

9640 `setlocale(LC_ALL, "C")`

9641 shall be executed at start-up.

9642 Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default
 9643 action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG_IGN) by
 9644 the calling process image shall be set to be ignored by the new process image. Signals set to be
 9645 caught by the calling process image shall be set to the default action in the new process image
 9646 (see **<signal.h>**). If the SIGCHLD signal is set to be ignored by the calling process image, it is
 9647 unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the new
 9648 XSI process image. After a successful call to any of the *exec* functions, alternate signal stacks are not
 9649 preserved and the SA_ONSTACK flag shall be cleared for all signals.

9650		After a successful call to any of the <i>exec</i> functions, any functions previously registered by <i>atexit()</i>
9651		are no longer registered.
9652	XSI	If the ST_NOSUID bit is set for the file system containing the new process image file, then the
9653		effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged
9654		in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is
9655		set, the effective user ID of the new process image shall be set to the user ID of the new process
9656		image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the
9657		effective group ID of the new process image shall be set to the group ID of the new process
9658		image file. The real user ID, real group ID, and supplementary group IDs of the new process
9659		image shall remain the same as those of the calling process image. The effective user ID and
9660		effective group ID of the new process image shall be saved (as the saved set-user-ID and the
9661		saved set-group-ID) for use by <i>setuid()</i> .
9662	XSI	Any shared memory segments attached to the calling process image shall not be attached to the
9663		new process image.
9664	SEM	Any named semaphores open in the calling process shall be closed as if by appropriate calls to
9665		<i>sem_close()</i> .
9666	TYM	Any blocks of typed memory that were mapped in the calling process are unmapped, as if
9667		<i>munmap()</i> was implicitly called to unmap them.
9668	ML	Memory locks established by the calling process via calls to <i>mlockall()</i> or <i>mlock()</i> shall be
9669		removed. If locked pages in the address space of the calling process are also mapped into the
9670		address spaces of other processes and are locked by those processes, the locks established by the
9671		other processes shall be unaffected by the call by this process to the <i>exec</i> function. If the <i>exec</i>
9672		function fails, the effect on memory locks is unspecified.
9673	MF SHM	Memory mappings created in the process are unmapped before the address space is rebuilt for
9674		the new process image.
9675	PS	For the SCHED_FIFO and SCHED_RR scheduling policies, the policy and priority settings shall
9676		not be changed by a call to an <i>exec</i> function. For other scheduling policies, the policy and priority
9677		settings on <i>exec</i> are implementation-defined.
9678	TMR	Per-process timers created by the calling process shall be deleted before replacing the current
9679		process image with the new process image.
9680	MSG	All open message queue descriptors in the calling process shall be closed, as described in
9681		<i>mq_close()</i> .
9682	AIO	Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O
9683		operations that are not canceled shall complete as if the <i>exec</i> function had not yet occurred, but
9684		any associated signal notifications shall be suppressed. It is unspecified whether the <i>exec</i>
9685		function itself blocks awaiting such I/O completion. In no event, however, shall the new process
9686		image created by the <i>exec</i> function be affected by the presence of outstanding asynchronous I/O
9687		operations at the time the <i>exec</i> function is called. Whether any I/O is canceled, and which I/O
9688		may be canceled upon <i>exec</i> , is implementation-defined.
9689	CPT	The new process image shall inherit the CPU-time clock of the calling process image. This
9690		inheritance means that the process CPU-time clock of the process being <i>exec</i> -ed shall not be
9691		reinitialized or altered as a result of the <i>exec</i> function other than to reflect the time spent by the
9692		process executing the <i>exec</i> function itself.
9693	TCT	The initial value of the CPU-time clock of the initial thread of the new process image shall be set
9694		to zero.

9695	TRC	If the calling process is being traced, the new process image shall continue to be traced into the same trace stream as the original process image, but the new process image shall not inherit the mapping of trace event names to trace event type identifiers that was defined by calls to the <i>posix_trace_eventid_open()</i> or the <i>posix_trace_trid_eventid_open()</i> functions in the calling process image.
9696		
9697		
9698		
9699		
9700		If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described in the <i>posix_trace_shutdown()</i> function.
9701		
9702		The new process shall inherit at least the following attributes from the calling process image:
9703	XSI	• Nice value (see <i>nice()</i>)
9704	XSI	• <i>semadj</i> values (see <i>semop()</i>)
9705		• Process ID
9706		• Parent process ID
9707		• Process group ID
9708		• Session membership
9709		• Real user ID
9710		• Real group ID
9711		• Supplementary group IDs
9712		• Time left until an alarm clock signal (see <i>alarm()</i>)
9713		• Current working directory
9714		• Root directory
9715		• File mode creation mask (see <i>umask()</i>)
9716	XSI	• File size limit (see <i>ulimit()</i>)
9717		• Process signal mask (see <i>sigprocmask()</i>)
9718		• Pending signal (see <i>sigpending()</i>)
9719		• <i>tms_utime</i> , <i>tms_stime</i> , <i>tms_cutime</i> , and <i>tms_cstime</i> (see <i>times()</i>)
9720	XSI	• Resource limits
9721	XSI	• Controlling terminal
9722	XSI	• Interval timers
9723		All other process attributes defined in this volume of IEEE Std 1003.1-2001 shall be the same in the new and old process images. The inheritance of process attributes not defined by this volume of IEEE Std 1003.1-2001 is implementation-defined.
9724		
9725		
9726		A call to any <i>exec</i> function from a process with more than one thread shall result in all threads being terminated and the new executable image being loaded and executed. No destructor functions shall be called.
9727		
9728		
9729		Upon successful completion, the <i>exec</i> functions shall mark for update the <i>st_atime</i> field of the file.
9730		If an <i>exec</i> function failed but was able to locate the process image file, whether the <i>st_atime</i> field is marked for update is unspecified. Should the <i>exec</i> function succeed, the process image file shall be considered to have been opened with <i>open()</i> . The corresponding <i>close()</i> shall be considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the <i>exec</i> functions, <i>posix_spawn()</i> , or <i>posix_spawnp()</i> .
9731		
9732		
9733		
9734		

9735 The *argv[]* and *envp[]* arrays of pointers and the strings to which those arrays point shall not be
 9736 modified by a call to one of the *exec* functions, except as a consequence of replacing the process
 9737 image.

9738 XSI The saved resource limits in the new process image are set to be a copy of the process'
 9739 corresponding hard and soft limits.

9740 RETURN VALUE

9741 If one of the *exec* functions returns to the calling process image, an error has occurred; the return
 9742 value shall be -1 , and *errno* shall be set to indicate the error.

9743 ERRORS

9744 The *exec* functions shall fail if:

9745 [E2BIG] The number of bytes used by the new process image's argument list and
 9746 environment list is greater than the system-imposed limit of {ARG_MAX}
 9747 bytes.

9748 [EACCES] Search permission is denied for a directory listed in the new process image
 9749 file's path prefix, or the new process image file denies execution permission,
 9750 or the new process image file is not a regular file and the implementation does
 9751 not support execution of files of its type.

9752 [EINVAL] The new process image file has the appropriate permission and has a
 9753 recognized executable binary format, but the system does not support
 9754 execution of a file with this format.

9755 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* or *file*
 9756 argument.

9757 [ENAMETOOLONG]
 9758 The length of the *path* or *file* arguments exceeds {PATH_MAX} or a pathname
 9759 component is longer than {NAME_MAX}.

9760 [ENOENT] A component of *path* or *file* does not name an existing file or *path* or *file* is an
 9761 empty string.

9762 [ENOTDIR] A component of the new process image file's path prefix is not a directory.

9763 The *exec* functions, except for *execlp()* and *execvp()*, shall fail if:

9764 [ENOEXEC] The new process image file has the appropriate access permission but has an
 9765 unrecognized format.

9766 The *exec* functions may fail if:

9767 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 9768 resolution of the *path* or *file* argument.

9769 [ENAMETOOLONG]
 9770 As a result of encountering a symbolic link in resolution of the *path* argument,
 9771 the length of the substituted pathname string exceeded {PATH_MAX}.

9772 [ENOMEM] The new process image requires more memory than is allowed by the
 9773 hardware or system-imposed memory management constraints.

9774 [ETXTBSY] The new process image file is a pure procedure (shared text) file that is
 9775 currently open for writing by some process.

9776 **EXAMPLES**9777 **Using execl()**

9778 The following example executes the *ls* command, specifying the pathname of the executable
9779 (**/bin/ls**) and using arguments supplied directly to the command to produce single-column
9780 output.

```
9781 #include <unistd.h>
9782 int ret;
9783 ...
9784 ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

9785 **Using execle()**

9786 The following example is similar to **Using execl()**. In addition, it specifies the environment for
9787 the new process image using the *env* argument.

```
9788 #include <unistd.h>
9789 int ret;
9790 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
9791 ...
9792 ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

9793 **Using execlp()**

9794 The following example searches for the location of the *ls* command among the directories
9795 specified by the *PATH* environment variable.

```
9796 #include <unistd.h>
9797 int ret;
9798 ...
9799 ret = execlp ("ls", "ls", "-l", (char *)0);
```

9800 **Using execv()**

9801 The following example passes arguments to the *ls* command in the *cmd* array.

```
9802 #include <unistd.h>
9803 int ret;
9804 char *cmd[] = { "ls", "-l", (char *)0 };
9805 ...
9806 ret = execv ("/bin/ls", cmd);
```

Using `execve()`

The following example passes arguments to the *ls* command in the *cmd* array, and specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execve ("/bin/ls", cmd, env);
```

Using `execvp()`

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable, and passes arguments to the *ls* command in the *cmd* array.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execvp ("ls", cmd);
```

APPLICATION USAGE

As the state of conversion descriptors and message catalog descriptors in the new process image is undefined, conforming applications should not rely on their use and should close them prior to calling one of the *exec* functions.

Applications that require other than the default POSIX locale should call *setlocale()* with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

Applications should not depend on file descriptors 0, 1, and 2 being closed after an *exec*. A future version may allow these file descriptors to be automatically opened for any process.

RATIONALE

Early proposals required that the value of *argc* passed to *main()* be “one or greater”. This was driven by the same requirement in drafts of the ISO C standard. In fact, historical implementations have passed a value of zero when no arguments are supplied to the caller of the *exec* functions. This requirement was removed from the ISO C standard and subsequently removed from this volume of IEEE Std 1003.1-2001 as well. The wording, in particular the use of the word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an application. In fact, this is good practice, since many existing applications reference *argv[0]* without first checking the value of *argc*.

The requirement on a Strictly Conforming POSIX Application also states that the value passed as the first argument be a filename associated with the process being started. Although some existing applications pass a pathname rather than a filename in some circumstances, a filename is more generally useful, since the common usage of *argv[0]* is in printing diagnostics. In some cases the filename passed is not the actual filename of the file; for example, many implementations of the *login* utility use a convention of prefixing a hyphen (‘-’) to the actual filename, which indicates to the command interpreter being invoked that it is a “login shell”.

Historically there have been two ways that implementations can *exec* shell scripts.

One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()* functions return an [ENOEXEC] error for any file not recognizable as executable, including a shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file to be a shell script and invoke a known command interpreter to interpret such files. This is now required by IEEE Std 1003.1-2001. These implementations of *execvp()* and *execlp()* only give the [ENOEXEC] error in the rare case of a problem with the command interpreter's executable file. Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp()* or *execvp()*, although implementations can still give it.

Another way that some historical implementations handle shell scripts is by recognizing the first two bytes of the file as the character string "#!" and using the remainder of the first line of the file as the name of the command interpreter to execute.

One potential source of confusion noted by the standard developers is over how the contents of a process image file affect the behavior of the *exec* family of functions. The following is a description of the actions taken:

1. If the process image file is a valid executable (in a format that is executable and valid and having appropriate permission) for this system, then the system executes the file.
2. If the process image file has appropriate permission and is in a format that is executable but not valid for this system (such as a recognized binary for another architecture), then this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
3. If the process image file has appropriate permission but is not otherwise recognized:
 - a. If this is a call to *execlp()* or *execvp()*, then they invoke a command interpreter assuming that the process image file is a shell script.
 - b. If this is not a call to *execlp()* or *execvp()*, then an error occurs and *errno* is set to [ENOEXEC].

Applications that do not require to access their arguments may use the form:

```
main(void)
```

as specified in the ISO C standard. However, the implementation will always provide the two arguments *argc* and *argv*, even if they are not used.

Some implementations provide a third argument to *main()* called *envp*. This is defined as a pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments, so implementations must support applications written this way. Since this volume of IEEE Std 1003.1-2001 defines the global variable *environ*, which is also provided by historical implementations and can be used anywhere that *envp* could be used, there is no functional need for the *envp* argument. Applications should use the *getenv()* function rather than accessing the environment directly via either *envp* or *environ*. Implementations are required to support the two-argument calling sequence, but this does not prohibit an implementation from supporting *envp* as an optional third argument.

This volume of IEEE Std 1003.1-2001 specifies that signals set to SIG_IGN remain set to SIG_IGN, and that the process signal mask be unchanged across an *exec*. This is consistent with historical implementations, and it permits some useful functionality, such as the *nohup* command. However, it should be noted that many existing applications wrongly assume that they start with certain signals set to the default action and/or unblocked. In particular, applications written with a simpler signal model that does not include blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked with some signals blocked. Therefore, it is best not to block or ignore signals across *execs* without explicit reason to do so,

and especially not to block signals across *execs* of arbitrary (not closely co-operating) programs.

The *exec* functions always save the value of the effective user ID and effective group ID of the process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of the process image file is set.

The statement about *argv[]* and *envp[]* being constants is included to make explicit to future writers of language bindings that these objects are completely constant. Due to a limitation of the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the natural choice, given that these functions do not modify either the array of pointers or the characters to which the function points, but this would disallow existing correct code. Instead, only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src* derived from the ISO C standard summarizes the compatibility:

<i>dst:</i>	char *[]	const char *[]	char *const[]	const char *const[]
<i>src:</i>				
char *[]	VALID	—	VALID	—
const char *[]	—	VALID	—	VALID
char *const[]	—	—	VALID	—
const char *const[]	—	—	—	VALID

Since all existing code has a source type matching the first row, the column that gives the most valid combinations is the third column. The only other possibility is the fourth column, but using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth column cannot be used, because the declaration a non-expert would naturally use would be that in the second row.

The ISO C standard and this volume of IEEE Std 1003.1-2001 do not conflict on the use of *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in the same way as an entry point (for example, *fork()*), it conforms to both standards. A library can contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence and no problem ensues. The situation is similar for *environ*: the definition in this volume of IEEE Std 1003.1-2001 is to be used if there is no user-provided *environ* to take precedence. At least three implementations are known to exist that solve this problem.

[E2BIG] The limit {ARG_MAX} applies not just to the size of the argument list, but to the sum of that and the size of the environment list.

[EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the new process image file is corrupted. They are non-conforming.

[EINVAL] This error condition was added to IEEE Std 1003.1-2001 to allow an implementation to detect executable files generated for different architectures, and indicate this situation to the application. Historical implementations of shells, *execvp()*, and *execlp()* that encounter an [ENOEXEC] error will execute a shell on the assumption that the file is a shell script. This will not produce the desired effect when the file is a valid executable for a different architecture. An implementation may now choose to avoid this problem by returning [EINVAL] when a valid executable for a different architecture is encountered. Some historical implementations return [EINVAL] to indicate that the *path* argument contains a character with the high order bit set. The standard developers chose to deviate from historical practice for the following reasons:

1. The new utilization of [EINVAL] will provide some measure of utility to the user community.
2. Historical use of [EINVAL] is not acceptable in an internationalized operating environment.

[ENAMETOOLONG]

Since the file pathname may be constructed by taking elements in the *PATH* variable and putting them together with the filename, the [ENAMETOOLONG] error condition could also be reached this way.

[ETXTBSY]

System V returns this error when the executable file is currently open for writing by some process. This volume of IEEE Std 1003.1-2001 neither requires nor prohibits this behavior.

Other systems (such as System V) may return [EINTR] from *exec*. This is not addressed by this volume of IEEE Std 1003.1-2001, but implementations may have a window between the call to *exec* and the time that a signal could cause one of the *exec* calls to return with [EINTR].

An explicit statement regarding the floating-point environment (as defined in the **<fenv.h>** header) was added to make it clear that the floating-point environment is set to its default when a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the default for other process and thread start-up functions is covered by more generic statements in their descriptions and can be summarized as follows:

posix_spawn() Set to default.

fork() Inherit.

pthread_create() Inherit.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), *atexit()*, *chmod()*, *close()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getitimer()*, *getrlimit()*, *mmap()*, *nice()*, *posix_spawn()*, *posix_trace_eventid_open()*, *posix_trace_shutdown()*, *posix_trace_trid_eventid_open()*, *putenv()*, *semop()*, *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*, *sigpending()*, *sigprocmask()*, *system()*, *times()*, *ulimit()*, *umask()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, **<unistd.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, behavior is defined for when the process image file is not a valid executable.
- In this issue, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective group ID of the new process image shall be saved (as the saved set-user-ID and the saved set-group-ID) for use by the *setuid()* function.

9987	• The [ELOOP] mandatory error condition is added.
9988	• A second [ENAMETOOLONG] is added as an optional error condition.
9989	• The [ETXTBSY] optional error condition is added.
9990	The following changes were made to align with the IEEE P1003.1a draft standard:
9991	• The [EINVAL] mandatory error condition is added.
9992	• The [ELOOP] optional error condition is added.
9993	The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.
9994	The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for
9995	typed memory.
9996	The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
9997	The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.
9998	IEEE PASC Interpretation 1003.1 #132 is applied.
9999	The DESCRIPTION is updated to make it explicit that the floating-point environment in the new
10000	process image is set to the default.
10001	The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents
10002	of a process image file affect the behavior of the <i>exec</i> functions.
10003	IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 15 is applied, adding a new paragraph to
10004	the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change
10005	addresses a security concern, where implementations may want to reopen file descriptors 0, 1,
10006	and 2 for programs with the set-user-id or set-group-id file mode bits calling the <i>exec</i> family of
10007	functions.

10008 **NAME**

10009 exit, _Exit, _exit — terminate a process

10010 **SYNOPSIS**

10011 #include <stdlib.h>

10012 void exit(int *status*);10013 void _Exit(int *status*);

10014 #include <unistd.h>

10015 void _exit(int *status*);10016 **DESCRIPTION**

10017 CX For *exit()* and *_Exit()*: The functionality described on this reference page is aligned with the
 10018 ISO C standard. Any conflict between the requirements described here and the ISO C standard is
 10019 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10020 CX The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only
 10021 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.

10022 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their
 10023 registration, except that a function is called after any previously registered functions that had
 10024 already been called at the time it was registered. Each function is called as many times as it was
 10025 registered. If, during the call to any such function, a call to the *longjmp()* function is made that
 10026 would terminate the call to the registered function, the behavior is undefined.

10027 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall
 10028 not be called and the rest of the *exit()* processing shall not be completed. If *exit()* is called more
 10029 than once, the behavior is undefined.

10030 The *exit()* function shall then flush all open streams with unwritten buffered data, close all open
 10031 streams, and remove all files created by *tmpfile()*. Finally, control shall be terminated with the
 10032 consequences described below.

10033 CX The *_Exit()* and *_exit()* functions shall be functionally equivalent.

10034 CX The *_Exit()* and *_exit()* functions shall not call functions registered with *atexit()* nor any
 10035 registered signal handlers. Whether open streams are flushed or closed, or temporary files are
 10036 removed is implementation-defined. Finally, the calling process is terminated with the
 10037 consequences described below.

10038 CX These functions shall terminate the calling process with the following consequences:

10039 **Note:** These consequences are all extensions to the ISO C standard and are not further CX shaded.
 10040 However, XSI extensions are shaded.

10041 XSI • All of the file descriptors, directory streams, conversion descriptors, and message catalog
 10042 descriptors open in the calling process shall be closed.

10043 XSI • If the parent process of the calling process is executing a *wait()* or *waitpid()*, and has neither
 10044 set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, it shall be notified of the calling
 10045 process' termination and the low-order eight bits (that is, bits 0377) of *status* shall be made
 10046 available to it. If the parent is not waiting, the child's status shall be made available to it
 10047 when the parent subsequently executes *wait()* or *waitpid()*.

10048 XSI The semantics of the *waitid()* function shall be equivalent to *wait()*.

10049 XSI • If the parent process of the calling process is not executing a *wait()* or *waitpid()*, and has
 10050 neither set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, the calling process shall
 10051 be transformed into a *zombie process*. A *zombie process* is an inactive process and it shall be

10052		deleted at some later time when its parent process executes <code>wait()</code> or <code>waitpid()</code> .
10053	XSI	The semantics of the <code>waitid()</code> function shall be equivalent to <code>wait()</code> .
10054		• Termination of a process does not directly terminate its children. The sending of a <code>SIGHUP</code> signal as described below indirectly terminates children in some circumstances.
10055		
10056		• Either:
10057		If the implementation supports the <code>SIGCHLD</code> signal, a <code>SIGCHLD</code> shall be sent to the parent process.
10058		
10059		Or:
10060	XSI	If the parent process has set its <code>SA_NOCLDWAIT</code> flag, or set <code>SIGCHLD</code> to <code>SIG_IGN</code> , the status shall be discarded, and the lifetime of the calling process shall end immediately. If
10061		<code>SA_NOCLDWAIT</code> is set, it is implementation-defined whether a <code>SIGCHLD</code> signal is sent to
10062		the parent process.
10063		
10064		• The parent process ID of all of the calling process' existing child processes and zombie processes shall be set to the process ID of an implementation-defined system process. That is, these processes shall be inherited by a special system process.
10065		
10066		
10067	XSI	• Each attached shared-memory segment is detached and the value of <code>shm_nattch</code> (see
10068		<code>shmget()</code>) in the data structure associated with its shared memory ID shall be decremented by
10069		1.
10070	XSI	• For each semaphore for which the calling process has set a <code>semadj</code> value (see <code>semop()</code>), that
10071		value shall be added to the <code>semval</code> of the specified semaphore.
10072		• If the process is a controlling process, the <code>SIGHUP</code> signal shall be sent to each process in the
10073		foreground process group of the controlling terminal belonging to the calling process.
10074		• If the process is a controlling process, the controlling terminal associated with the session
10075		shall be disassociated from the session, allowing it to be acquired by a new controlling
10076		process.
10077		• If the exit of the process causes a process group to become orphaned, and if any member of
10078		the newly-orphaned process group is stopped, then a <code>SIGHUP</code> signal followed by a
10079		<code>SIGCONT</code> signal shall be sent to each process in the newly-orphaned process group.
10080	SEM	• All open named semaphores in the calling process shall be closed as if by appropriate calls to
10081		<code>sem_close()</code> .
10082	ML	• Any memory locks established by the process via calls to <code>mlockall()</code> or <code>mlock()</code> shall be
10083		removed. If locked pages in the address space of the calling process are also mapped into the
10084		address spaces of other processes and are locked by those processes, the locks established by
10085		the other processes shall be unaffected by the call by this process to <code>_Exit()</code> or <code>_exit()</code> .
10086	MF SHM	• Memory mappings that were created in the process shall be unmapped before the process is
10087		destroyed.
10088	TYM	• Any blocks of typed memory that were mapped in the calling process shall be unmapped, as
10089		if <code>munmap()</code> was implicitly called to unmap them.
10090	MSG	• All open message queue descriptors in the calling process shall be closed as if by appropriate
10091		calls to <code>mq_close()</code> .
10092	AIO	• Any outstanding cancelable asynchronous I/O operations may be canceled. Those
10093		asynchronous I/O operations that are not canceled shall complete as if the <code>_Exit()</code> or <code>_exit()</code>
10094		operation had not yet occurred, but any associated signal notifications shall be suppressed.

10095 The `_Exit()` or `_exit()` operation may block awaiting such I/O completion. Whether any I/O
 10096 is canceled, and which I/O may be canceled upon `_Exit()` or `_exit()`, is implementation-
 10097 defined.

10098 • Threads terminated by a call to `_Exit()` or `_exit()` shall not invoke their cancellation cleanup
 10099 handlers or per-thread data destructors.

10100 TRC • If the calling process is a trace controller process, any trace streams that were created by the
 10101 calling process shall be shut down as described by the `posix_trace_shutdown()` function, and
 10102 any process' mapping of trace event names to trace event type identifiers built for these trace
 10103 streams may be deallocated.

10104 RETURN VALUE

10105 These functions do not return.

10106 ERRORS

10107 No errors are defined.

10108 EXAMPLES

10109 None.

10110 APPLICATION USAGE

10111 Normally applications should use `exit()` rather than `_Exit()` or `_exit()`.

10112 RATIONALE

10113 Process Termination

10114 Early proposals drew a distinction between normal and abnormal process termination.
 10115 Abnormal termination was caused only by certain signals and resulted in implementation-
 10116 defined “actions”, as discussed below. Subsequent proposals distinguished three types of
 10117 termination: *normal termination* (as in the current specification), *simple abnormal termination*, and
 10118 *abnormal termination with actions*. Again the distinction between the two types of abnormal
 10119 termination was that they were caused by different signals and that implementation-defined
 10120 actions would result in the latter case. Given that these actions were completely
 10121 implementation-defined, the early proposals were only saying when the actions could occur and
 10122 how their occurrence could be detected, but not what they were. This was of little or no use to
 10123 conforming applications, and thus the distinction is not made in this volume of
 10124 IEEE Std 1003.1-2001.

10125 The implementation-defined actions usually include, in most historical implementations, the
 10126 creation of a file named **core** in the current working directory of the process. This file contains an
 10127 image of the memory of the process, together with descriptive information about the process,
 10128 perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

10129 There is a potential security problem in creating a **core** file if the process was set-user-ID and the
 10130 current user is not the owner of the program, if the process was set-group-ID and none of the
 10131 user's groups match the group of the program, or if the user does not have permission to write in
 10132 the current directory. In this situation, an implementation either should not create a **core** file or
 10133 should make it unreadable by the user.

10134 Despite the silence of this volume of IEEE Std 1003.1-2001 on this feature, applications are
 10135 advised not to create files named **core** because of potential conflicts in many implementations.
 10136 Some implementations use a name other than **core** for the file; for example, by appending the
 10137 process ID to the filename.

Terminating a Process

It is important that the consequences of process termination as described occur regardless of whether the process called `_exit()` (perhaps indirectly through `exit()`) or instead was terminated due to a signal or for some other reason. Note that in the specific case of `exit()` this means that the *status* argument to `exit()` is treated in the same way as the *status* argument to `_exit()`.

A language other than C may have other termination primitives than the C-language `exit()` function, and programs written in such a language should use its native termination primitives, but those should have as part of their function the behavior of `_exit()` as described. Implementations in languages other than C are outside the scope of this version of this volume of IEEE Std 1003.1-2001, however.

As required by the ISO C standard, using **return** from `main()` has the same behavior (other than with respect to language scope issues) as calling `exit()` with the returned value. Reaching the end of the `main()` function has the same behavior as calling `exit(0)`.

A value of zero (or `EXIT_SUCCESS`, which is required to be zero) for the argument *status* conventionally indicates successful termination. This corresponds to the specification for `exit()` in the ISO C standard. The convention is followed by utilities such as *make* and various shells, which interpret a zero status from a child process as success. For this reason, applications should not call `exit(0)` or `_exit(0)` when they terminate unsuccessfully; for example, in signal-catching functions.

Historically, the implementation-defined process that inherits children whose parents have terminated without waiting on them is called *init* and has a process ID of 1.

The sending of a `SIGHUP` to the foreground process group when a controlling process terminates corresponds to somewhat different historical implementations. In System V, the kernel sends a `SIGHUP` on termination of (essentially) a controlling process. In 4.2 BSD, the kernel does not send `SIGHUP` in a case like this, but the termination of a controlling process is usually noticed by a system daemon, which arranges to send a `SIGHUP` to the foreground process group with the `vhangup()` function. However, in 4.2 BSD, due to the behavior of the shells that support job control, the controlling process is usually a shell with no other processes in its process group. Thus, a change to make `_exit()` behave this way in such systems should not cause problems with existing applications.

The termination of a process may cause a process group to become orphaned in either of two ways. The connection of a process group to its parent(s) outside of the group depends on both the parents and their children. Thus, a process group may be orphaned by the termination of the last connecting parent process outside of the group or by the termination of the last direct descendant of the parent process(es). In either case, if the termination of a process causes a process group to become orphaned, processes within the group are disconnected from their job control shell, which no longer has any information on the existence of the process group. Stopped processes within the group would languish forever. In order to avoid this problem, newly orphaned process groups that contain stopped processes are sent a `SIGHUP` signal and a `SIGCONT` signal to indicate that they have been disconnected from their session. The `SIGHUP` signal causes the process group members to terminate unless they are catching or ignoring `SIGHUP`. Under most circumstances, all of the members of the process group are stopped if any of them are stopped.

The action of sending a `SIGHUP` and a `SIGCONT` signal to members of a newly orphaned process group is similar to the action of 4.2 BSD, which sends `SIGHUP` and `SIGCONT` to each stopped child of an exiting process. If such children exit in response to the `SIGHUP`, any additional descendants receive similar treatment at that time. In this volume of IEEE Std 1003.1-2001, the signals are sent to the entire process group at the same time. Also, in

10186 this volume of IEEE Std 1003.1-2001, but not in 4.2 BSD, stopped processes may be orphaned, but
 10187 may be members of a process group that is not orphaned; therefore, the action taken at `_exit()`
 10188 must consider processes other than child processes.

10189 It is possible for a process group to be orphaned by a call to `setpgid()` or `setsid()`, as well as by
 10190 process termination. This volume of IEEE Std 1003.1-2001 does not require sending `SIGHUP` and
 10191 `SIGCONT` in those cases, because, unlike process termination, those cases are not caused
 10192 accidentally by applications that are unaware of job control. An implementation can choose to
 10193 send `SIGHUP` and `SIGCONT` in those cases as an extension; such an extension must be
 10194 documented as required in **<signal.h>**.

10195 The ISO/IEC 9899:1999 standard adds the `_Exit()` function that results in immediate program
 10196 termination without triggering signals or `atexit()`-registered functions. In IEEE Std 1003.1-2001,
 10197 this is equivalent to the `_exit()` function.

10198 **FUTURE DIRECTIONS**

10199 None.

10200 **SEE ALSO**

10201 `atexit()`, `close()`, `fclose()`, `longjmp()`, `posix_trace_shutdown()`, `posix_trace_trid_eventid_open()`,
 10202 `semop()`, `shmget()`, `sigaction()`, `wait()`, `waitid()`, `waitpid()`, the Base Definitions volume of
 10203 IEEE Std 1003.1-2001, **<stdlib.h>**, **<unistd.h>**

10204 **CHANGE HISTORY**

10205 First released in Issue 1. Derived from Issue 1 of the SVID.

10206 **Issue 5**

10207 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 10208 Threads Extension.

10209 Interactions with the `SA_NOCLDWAIT` flag and `SIGCHLD` signal are further clarified.

10210 The values of `status` from `exit()` are better described.

10211 **Issue 6**

10212 Extensions beyond the ISO C standard are marked.

10213 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for
 10214 typed memory.

10215 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 10216 • The `_Exit()` function is included.
- 10217 • The DESCRIPTION is updated.

10218 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

10219 References to the `wait3()` function are removed.

10220 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the
 10221 DESCRIPTION.

10222 **NAME**

10223 exp, expf, expl — exponential function

10224 **SYNOPSIS**

10225 #include <math.h>

10226 double exp(double x);

10227 float expf(float x);

10228 long double expl(long double x);

10229 **DESCRIPTION**

10230 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 10231 conflict between the requirements described here and the ISO C standard is unintentional. This
 10232 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10233 These functions shall compute the base-*e* exponential of *x*.

10234 An application wishing to check for error situations should set *errno* to zero and call
 10235 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 10236 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 10237 zero, an error has occurred.

10238 **RETURN VALUE**10239 Upon successful completion, these functions shall return the exponential value of *x*.

10240 If the correct value would cause overflow, a range error shall occur and *exp()*, *expf()*, and *expl()*
 10241 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

10242 If the correct value would cause underflow, and is not representable, a range error may occur,
 10243 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

10244 **MX** If *x* is NaN, a NaN shall be returned.10245 If *x* is ± 0 , 1 shall be returned.10246 If *x* is $-\text{Inf}$, +0 shall be returned.10247 If *x* is $+\text{Inf}$, *x* shall be returned.

10248 If the correct value would cause underflow, and is representable, a range error may occur and
 10249 the correct value shall be returned.

10250 **ERRORS**

10251 These functions shall fail if:

10252 Range Error The result overflows.

10253 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10254 then *errno* shall be set to [ERANGE]. If the integer expression
 10255 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 10256 floating-point exception shall be raised.

10257 These functions may fail if:

10258 Range Error The result underflows.

10259 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10260 then *errno* shall be set to [ERANGE]. If the integer expression
 10261 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 10262 floating-point exception shall be raised.

10263 **EXAMPLES**

10264 None.

10265 **APPLICATION USAGE**

10266 Note that for IEEE Std 754-1985 **double**, $709.8 < x$ implies $\exp(x)$ has overflowed. The value
10267 $x < -708.4$ implies $\exp(x)$ has underflowed.

10268 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`
10269 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

10270 **RATIONALE**

10271 None.

10272 **FUTURE DIRECTIONS**

10273 None.

10274 **SEE ALSO**

10275 `feclearexcept()`, `fetestexcept()`, `isnan()`, `log()`, the Base Definitions volume of IEEE Std 1003.1-2001,
10276 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

10277 **CHANGE HISTORY**

10278 First released in Issue 1. Derived from Issue 1 of the SVID.

10279 **Issue 5**

10280 The DESCRIPTION is updated to indicate how an application should check for an error. This
10281 text was previously published in the APPLICATION USAGE section.

10282 **Issue 6**10283 The `expf()` and `expl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

10284 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
10285 revised to align with the ISO/IEC 9899:1999 standard.

10286 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
10287 marked.

10288 **NAME**

10289 exp2, exp2f, exp2l — exponential base 2 functions

10290 **SYNOPSIS**

10291 #include <math.h>

10292 double exp2(double x);

10293 float exp2f(float x);

10294 long double exp2l(long double x);

10295 **DESCRIPTION**

10296 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 10297 conflict between the requirements described here and the ISO C standard is unintentional. This
 10298 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10299 These functions shall compute the base-2 exponential of x .

10300 An application wishing to check for error situations should set *errno* to zero and call
 10301 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 10302 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 10303 zero, an error has occurred.

10304 **RETURN VALUE**10305 Upon successful completion, these functions shall return 2^x .

10306 If the correct value would cause overflow, a range error shall occur and *exp2()*, *exp2f()*, and
 10307 *exp2l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 10308 respectively.

10309 If the correct value would cause underflow, and is not representable, a range error may occur,
 10310 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

10311 MX If x is NaN, a NaN shall be returned.10312 If x is ± 0 , 1 shall be returned.10313 If x is $-\text{Inf}$, +0 shall be returned.10314 If x is $+\text{Inf}$, x shall be returned.

10315 If the correct value would cause underflow, and is representable, a range error may occur and
 10316 the correct value shall be returned.

10317 **ERRORS**

10318 These functions shall fail if:

10319 Range Error The result overflows.

10320 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10321 then *errno* shall be set to [ERANGE]. If the integer expression
 10322 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 10323 floating-point exception shall be raised.

10324 These functions may fail if:

10325 Range Error The result underflows.

10326 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10327 then *errno* shall be set to [ERANGE]. If the integer expression
 10328 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 10329 floating-point exception shall be raised.

10330 **EXAMPLES**

10331 None.

10332 **APPLICATION USAGE**

10333 For IEEE Std 754-1985 **double**, $1024 \leq x$ implies $\exp2(x)$ has overflowed. The value $x < -1022$
10334 implies $\exp(x)$ has underflowed.

10335 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`
10336 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

10337 **RATIONALE**

10338 None.

10339 **FUTURE DIRECTIONS**

10340 None.

10341 **SEE ALSO**

10342 `exp()`, `feclearexcept()`, `fetestexcept()`, `isnan()`, `log()`, the Base Definitions volume of
10343 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
10344 <math.h>

10345 **CHANGE HISTORY**

10346 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

10347 **NAME**

10348 expm1, expm1f, expm1l — compute exponential functions

10349 **SYNOPSIS**

10350 #include <math.h>

10351 double expm1(double x);

10352 float expm1f(float x);

10353 long double expm1l(long double x);

10354 **DESCRIPTION**

10355 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 10356 conflict between the requirements described here and the ISO C standard is unintentional. This
 10357 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10358 These functions shall compute $e^x - 1.0$.

10359 An application wishing to check for error situations should set *errno* to zero and call
 10360 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 10361 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 10362 zero, an error has occurred.

10363 **RETURN VALUE**10364 Upon successful completion, these functions return $e^x - 1.0$.

10365 If the correct value would cause overflow, a range error shall occur and *expm1()*, *expm1f()*, and
 10366 *expm1l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 10367 respectively.

10368 **MX** If *x* is NaN, a NaN shall be returned.10369 If *x* is ± 0 , ± 0 shall be returned.10370 If *x* is $-\text{Inf}$, -1 shall be returned.10371 If *x* is $+\text{Inf}$, *x* shall be returned.10372 If *x* is subnormal, a range error may occur and *x* should be returned.10373 **ERRORS**

10374 These functions shall fail if:

10375 Range Error The result overflows.

10376 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10377 then *errno* shall be set to [ERANGE]. If the integer expression
 10378 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 10379 floating-point exception shall be raised.

10380 These functions may fail if:

10381 **MX** Range Error The value of *x* is subnormal.

10382 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10383 then *errno* shall be set to [ERANGE]. If the integer expression
 10384 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 10385 floating-point exception shall be raised.

10386 **EXAMPLES**

10387 None.

10388 **APPLICATION USAGE**10389 The value of *expm1*(*x*) may be more accurate than *exp*(*x*)−1.0 for small values of *x*.10390 The *expm1*() and *log1p*() functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:10391 $\text{expm1}(n * \text{log1p}(x)) / x$ 10392 when *x* is very small (for example, when calculating small daily interest rates). These functions
10393 also simplify writing accurate inverse hyperbolic functions.10394 For IEEE Std 754-1985 **double**, $709.8 < x$ implies *expm1*(*x*) has overflowed.10395 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
10396 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.10397 **RATIONALE**

10398 None.

10399 **FUTURE DIRECTIONS**

10400 None.

10401 **SEE ALSO**10402 *exp*(), *feclearexcept*(), *fetestexcept*(), *ilogb*(), *log1p*(), the Base Definitions volume of
10403 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
10404 <math.h>10405 **CHANGE HISTORY**

10406 First released in Issue 4, Version 2.

10407 **Issue 5**

10408 Moved from X/OPEN UNIX extension to BASE.

10409 **Issue 6**10410 The *expm1f*() and *expm1l*() functions are added for alignment with the ISO/IEC 9899:1999
10411 standard.10412 The *expm1*() function is no longer marked as an extension.10413 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
10414 revised to align with the ISO/IEC 9899:1999 standard.10415 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
10416 marked.

10417 **NAME**

10418 fabs, fabsf, fabsl — absolute value function

10419 **SYNOPSIS**

10420 #include <math.h>

10421 double fabs(double x);

10422 float fabsf(float x);

10423 long double fabsl(long double x);

10424 **DESCRIPTION**

10425 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 10426 conflict between the requirements described here and the ISO C standard is unintentional. This
 10427 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10428 These functions shall compute the absolute value of their argument x , $|x|$.10429 **RETURN VALUE**10430 Upon successful completion, these functions shall return the absolute value of x .10431 MX If x is NaN, a NaN shall be returned.10432 If x is ± 0 , $+0$ shall be returned.10433 If x is $\pm \text{Inf}$, $+\text{Inf}$ shall be returned.10434 **ERRORS**

10435 No errors are defined.

10436 **EXAMPLES**

10437 None.

10438 **APPLICATION USAGE**

10439 None.

10440 **RATIONALE**

10441 None.

10442 **FUTURE DIRECTIONS**

10443 None.

10444 **SEE ALSO**10445 *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>10446 **CHANGE HISTORY**

10447 First released in Issue 1. Derived from Issue 1 of the SVID.

10448 **Issue 5**

10449 The DESCRIPTION is updated to indicate how an application should check for an error. This
 10450 text was previously published in the APPLICATION USAGE section.

10451 **Issue 6**10452 The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

10453 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 10454 revised to align with the ISO/IEC 9899:1999 standard.

10455 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 10456 marked.

10457 **NAME**

10458 fattach — attach a STREAMS-based file descriptor to a file in the file system name space
 10459 **(STREAMS)**

10460 **SYNOPSIS**

```
10461 xSR       #include <stropts.h>
10462       int fattach(int fildev, const char *path);
10463
```

10464 **DESCRIPTION**

10465 The *fattach()* function shall attach a STREAMS-based file descriptor to a file, effectively
 10466 associating a pathname with *fildev*. The application shall ensure that the *fildev* argument is a
 10467 valid open file descriptor associated with a STREAMS file. The *path* argument points to a
 10468 pathname of an existing file. The application shall have the appropriate privileges or be the
 10469 owner of the file named by *path* and have write permission. A successful call to *fattach()* shall
 10470 cause all pathnames that name the file named by *path* to name the STREAMS file associated with
 10471 *fildev*, until the STREAMS file is detached from the file. A STREAMS file can be attached to more
 10472 than one file and can have several pathnames associated with it.

10473 The attributes of the named STREAMS file shall be initialized as follows: the permissions, user
 10474 ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1,
 10475 and the size and device identifier are set to those of the STREAMS file associated with *fildev*. If
 10476 any attributes of the named STREAMS file are subsequently changed (for example, by *chmod()*),
 10477 neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildev*
 10478 refers shall be affected.

10479 File descriptors referring to the underlying file, opened prior to an *fattach()* call, shall continue to
 10480 refer to the underlying file.

10481 **RETURN VALUE**

10482 Upon successful completion, *fattach()* shall return 0. Otherwise, *-1* shall be returned and *errno*
 10483 set to indicate the error.

10484 **ERRORS**

10485 The *fattach()* function shall fail if:

10486	[EACCES]	Search permission is denied for a component of the path prefix, or the process is the owner of <i>path</i> but does not have write permissions on the file named by <i>path</i> .
10487		
10488		
10489	[EBADF]	The <i>fildev</i> argument is not a valid open file descriptor.
10490	[EBUSY]	The file named by <i>path</i> is currently a mount point or has a STREAMS file attached to it.
10491		
10492	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
10493		
10494	[ENAMETOOLONG]	
10495		The size of <i>path</i> exceeds {PATH_MAX} or a component of <i>path</i> is longer than {NAME_MAX}.
10496		
10497	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
10498	[ENOTDIR]	A component of the path prefix is not a directory.
10499	[EPERM]	The effective user ID of the process is not the owner of the file named by <i>path</i> and the process does not have appropriate privilege.
10500		

10501 The *fattach()* function may fail if:

10502 [EINVAL] The *fildev* argument does not refer to a STREAMS file.

10503 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
10504 resolution of the *path* argument.

10505 [ENAMETOOLONG]

10506 Pathname resolution of a symbolic link produced an intermediate result
10507 whose length exceeds {PATH_MAX}.

10508 [EXDEV] A link to a file on another file system was attempted.

10509 EXAMPLES

10510 Attaching a File Descriptor to a File

10511 In the following example, *fd* refers to an open STREAMS file. The call to *fattach()* associates this
10512 STREAM with the file **/tmp/named-STREAM**, such that any future calls to open **/tmp/named-**
10513 **STREAM**, prior to breaking the attachment via a call to *fdetach()*, will instead create a new file
10514 handle referring to the STREAMS file associated with *fd*.

```
10515 #include <stropts.h>
```

```
10516 ...
```

```
10517     int fd;
```

```
10518     char *filename = "/tmp/named-STREAM";
```

```
10519     int ret;
```

```
10520     ret = fattach(fd, filename);
```

10521 APPLICATION USAGE

10522 The *fattach()* function behaves similarly to the traditional *mount()* function in the way a file is
10523 temporarily replaced by the root directory of the mounted file system. In the case of *fattach()*, the
10524 replaced file need not be a directory and the replacing file is a STREAMS file.

10525 RATIONALE

10526 The file attributes of a file which has been the subject of an *fattach()* call are specifically set
10527 because of an artefact of the original implementation. The internal mechanism was the same as
10528 for the *mount()* function. Since *mount()* is typically only applied to directories, the effects when
10529 applied to a regular file are a little surprising, especially as regards the link count which rigidly
10530 remains one, even if there were several links originally and despite the fact that all original links
10531 refer to the STREAM as long as the *fattach()* remains in effect.

10532 FUTURE DIRECTIONS

10533 None.

10534 SEE ALSO

10535 *fdetach()*, *isastream()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stropts.h>**

10536 CHANGE HISTORY

10537 First released in Issue 4, Version 2.

10538 Issue 5

10539 Moved from X/OPEN UNIX extension to BASE.

10540 The [EXDEV] error is added to the list of optional errors in the ERRORS section.

10541 **Issue 6**

10542 This function is marked as part of the XSI STREAMS Option Group.

10543 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

10544 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
10545 [ELOOP] error condition is added.

10546 **NAME**

10547 fchdir — change working directory

10548 **SYNOPSIS**

10549 xSI #include <unistd.h>

10550 int fchdir(int *fildev*);

10551

10552 **DESCRIPTION**

10553 The *fchdir()* function shall be equivalent to *chdir()* except that the directory that is to be the new
 10554 current working directory is specified by the file descriptor *fildev*.

10555 A conforming application can obtain a file descriptor for a file of type directory using *open()*,
 10556 provided that the file status flags and access modes do not contain O_WRONLY or O_RDWR.

10557 **RETURN VALUE**

10558 Upon successful completion, *fchdir()* shall return 0. Otherwise, it shall return -1 and set *errno* to
 10559 indicate the error. On failure the current working directory shall remain unchanged.

10560 **ERRORS**10561 The *fchdir()* function shall fail if:

10562 [EACCES] Search permission is denied for the directory referenced by *fildev*.

10563 [EBADF] The *fildev* argument is not an open file descriptor.

10564 [ENOTDIR] The open file descriptor *fildev* does not refer to a directory.

10565 The *fchdir()* may fail if:

10566 [EINTR] A signal was caught during the execution of *fchdir()*.

10567 [EIO] An I/O error occurred while reading from or writing to the file system.

10568 **EXAMPLES**

10569 None.

10570 **APPLICATION USAGE**

10571 None.

10572 **RATIONALE**

10573 None.

10574 **FUTURE DIRECTIONS**

10575 None.

10576 **SEE ALSO**10577 *chdir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>10578 **CHANGE HISTORY**

10579 First released in Issue 4, Version 2.

10580 **Issue 5**

10581 Moved from X/OPEN UNIX extension to BASE.

10582 **NAME**

10583 fchmod — change mode of a file

10584 **SYNOPSIS**

10585 #include <sys/stat.h>

10586 int fchmod(int *fildes*, mode_t *mode*);10587 **DESCRIPTION**

10588 The *fchmod()* function shall be equivalent to *chmod()* except that the file whose permissions are
 10589 changed is specified by the file descriptor *fildes*.

10590 SHM If *fildes* references a shared memory object, the *fchmod()* function need only affect the S_IRUSR,
 10591 S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.

10592 TYM If *fildes* references a typed memory object, the behavior of *fchmod()* is unspecified.

10593 If *fildes* refers to a socket, the behavior of *fchmod()* is unspecified.

10594 XSR If *fildes* refers to a STREAM (which is *fattach()*-ed into the file system name space) the call
 10595 returns successfully, doing nothing.

10596 **RETURN VALUE**

10597 Upon successful completion, *fchmod()* shall return 0. Otherwise, it shall return -1 and set *errno* to
 10598 indicate the error.

10599 **ERRORS**

10600 The *fchmod()* function shall fail if:

10601 [EBADF] The *fildes* argument is not an open file descriptor.

10602 [EPERM] The effective user ID does not match the owner of the file and the process
 10603 does not have appropriate privilege.

10604 [EROFS] The file referred to by *fildes* resides on a read-only file system.

10605 The *fchmod()* function may fail if:

10606 XSI [EINTR] The *fchmod()* function was interrupted by a signal.

10607 XSI [EINVAL] The value of the *mode* argument is invalid.

10608 [EINVAL] The *fildes* argument refers to a pipe and the implementation disallows
 10609 execution of *fchmod()* on a pipe.

10610 **EXAMPLES**10611 **Changing the Current Permissions for a File**

10612 The following example shows how to change the permissions for a file named **/home/cnd/mod1**
 10613 so that the owner and group have read/write/execute permissions, but the world only has
 10614 read/write permissions.

10615 #include <sys/stat.h>

10616 #include <fcntl.h>

10617 mode_t mode;

10618 int fildes;

10619 ...

10620 fildes = open("/home/cnd/mod1", O_RDWR);

10621 fchmod(fildes, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);

10622 APPLICATION USAGE

10623 None.

10624 RATIONALE

10625 None.

10626 FUTURE DIRECTIONS

10627 None.

10628 SEE ALSO

10629 *chmod()*, *chown()*, *creat()*, *fcntl()*, *fstatvfs()*, *mknod()*, *open()*, *read()*, *stat()*, *write()*, the Base
10630 Definitions volume of IEEE Std 1003.1-2001, <**sys/stat.h**>

10631 CHANGE HISTORY

10632 First released in Issue 4, Version 2.

10633 Issue 5

10634 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX
10635 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a
10636 second instance of [EINVAL] is defined in the list of optional errors.

10637 Issue 6

10638 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod()*
10639 behavior is unspecified for typed memory objects.

10640 **NAME**

10641 fchown — change owner and group of a file

10642 **SYNOPSIS**

10643 #include <unistd.h>

10644 int fchown(int *fildev*, uid_t *owner*, gid_t *group*);10645 **DESCRIPTION**

10646 The *fchown()* function shall be equivalent to *chown()* except that the file whose owner and group
 10647 are changed is specified by the file descriptor *fildev*.

10648 **RETURN VALUE**

10649 Upon successful completion, *fchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to
 10650 indicate the error.

10651 **ERRORS**10652 The *fchown()* function shall fail if:10653 [EBADF] The *fildev* argument is not an open file descriptor.

10654 [EPERM] The effective user ID does not match the owner of the file or the process does
 10655 not have appropriate privilege and _POSIX_CHOWN_RESTRICTED indicates
 10656 that such privilege is required.

10657 [EROFS] The file referred to by *fildev* resides on a read-only file system.10658 The *fchown()* function may fail if:

10659 [EINVAL] The owner or group ID is not a value supported by the implementation. The
 10660 XSR *fildev* argument refers to a pipe or socket or an *fattach()*-ed STREAM and the
 10661 implementation disallows execution of *fchown()* on a pipe.

10662 [EIO] A physical I/O error has occurred.

10663 [EINTR] The *fchown()* function was interrupted by a signal which was caught.10664 **EXAMPLES**10665 **Changing the Current Owner of a File**

10666 The following example shows how to change the owner of a file named */home/cnd/mod1* to
 10667 “jones” and the group to “cnd”.

10668 The numeric value for the user ID is obtained by extracting the user ID from the user database
 10669 entry associated with “jones”. Similarly, the numeric value for the group ID is obtained by
 10670 extracting the group ID from the group database entry associated with “cnd”. This example
 10671 assumes the calling program has appropriate privileges.

10672 #include <sys/types.h>

10673 #include <unistd.h>

10674 #include <fcntl.h>

10675 #include <pwd.h>

10676 #include <grp.h>

10677 struct passwd *pwd;

10678 struct group *grp;

10679 int fildev;

10680 ...

10681 fildev = open("/home/cnd/mod1", O_RDWR);

10682 pwd = getpwnam("jones");

```
10683     grp = getgrnam("cnd");
10684     fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

10685 APPLICATION USAGE

10686 None.

10687 RATIONALE

10688 None.

10689 FUTURE DIRECTIONS

10690 None.

10691 SEE ALSO

10692 *chown()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

10693 CHANGE HISTORY

10694 First released in Issue 4, Version 2.

10695 Issue 5

10696 Moved from X/OPEN UNIX extension to BASE.

10697 Issue 6

10698 The following changes were made to align with the IEEE P1003.1a draft standard:

- 10699 • Clarification is added that a call to *fchown()* may not be allowed on a pipe.

10700 The *fchown()* function is defined as mandatory.

10701 **NAME**

10702 fclose — close a stream

10703 **SYNOPSIS**

10704 #include <stdio.h>

10705 int fclose(FILE *stream);

10706 **DESCRIPTION**

10707 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 10708 conflict between the requirements described here and the ISO C standard is unintentional. This
 10709 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10710 The *fclose()* function shall cause the stream pointed to by *stream* to be flushed and the associated
 10711 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any
 10712 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be
 10713 disassociated from the file and any buffer set by the *setbuf()* or *setvbuf()* function shall be
 10714 disassociated from the stream. If the associated buffer was automatically allocated, it shall be
 10715 deallocated.

10716 CX The *fclose()* function shall mark for update the *st_ctime* and *st_mtime* fields of the underlying file,
 10717 if the stream was writable, and if buffered data remains that has not yet been written to the file.
 10718 The *fclose()* function shall perform the equivalent of a *close()* on the file descriptor that is
 10719 associated with the stream pointed to by *stream*.

10720 After the call to *fclose()*, any use of *stream* results in undefined behavior.

10721 **RETURN VALUE**

10722 CX Upon successful completion, *fclose()* shall return 0; otherwise, it shall return EOF and set *errno* to
 10723 indicate the error.

10724 **ERRORS**

10725 The *fclose()* function shall fail if:

10726 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 10727 process would be delayed in the write operation.

10728 CX [EBADF] The file descriptor underlying stream is not valid.

10729 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

10730 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

10731 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 10732 offset maximum associated with the corresponding stream.

10733 CX [EINTR] The *fclose()* function was interrupted by a signal.

10734 CX [EIO] The process is a member of a background process group attempting to write
 10735 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 10736 blocking SIGTTOU, and the process group of the process is orphaned. This
 10737 error may also be returned under implementation-defined conditions.

10738 CX [ENOSPC] There was no free space remaining on the device containing the file.

10739 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 10740 any process. A SIGPIPE signal shall also be sent to the thread.

10741 The *fclose()* function may fail if:

10742 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 10743 capabilities of the device.

10744 **EXAMPLES**

10745 None.

10746 **APPLICATION USAGE**

10747 None.

10748 **RATIONALE**

10749 None.

10750 **FUTURE DIRECTIONS**

10751 None.

10752 **SEE ALSO**

10753 *close()*, *fopen()*, *getrlimit()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-2001,
10754 <stdio.h>

10755 **CHANGE HISTORY**

10756 First released in Issue 1. Derived from Issue 1 of the SVID.

10757 **Issue 5**

10758 Large File Summit extensions are added.

10759 **Issue 6**

10760 Extensions beyond the ISO C standard are marked.

10761 The following new requirements on POSIX implementations derive from alignment with the
10762 Single UNIX Specification:

- 10763 • The [EFBIG] error is added as part of the large file support extensions.
- 10764 • The [ENXIO] optional error condition is added.

10765 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether
10766 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

10767 **NAME**

10768 fcntl — file control

10769 **SYNOPSIS**

10770 OH #include <unistd.h>

10771 #include <fcntl.h>

10772 int fcntl(int *fildes*, int *cmd*, ...);10773 **DESCRIPTION**

10774 The *fcntl()* function shall perform the operations described below on open files. The *fildes*
 10775 argument is a file descriptor.

10776 The available values for *cmd* are defined in <fcntl.h> and are as follows:

10777 **F_DUPFD** Return a new file descriptor which shall be the lowest numbered available
 10778 (that is, not already open) file descriptor greater than or equal to the third
 10779 argument, *arg*, taken as an integer of type **int**. The new file descriptor shall
 10780 refer to the same open file description as the original file descriptor, and shall
 10781 share any locks. The FD_CLOEXEC flag associated with the new file
 10782 descriptor shall be cleared to keep the file open across calls to one of the *exec*
 10783 functions.

10784 **F_GETFD** Get the file descriptor flags defined in <fcntl.h> that are associated with the
 10785 file descriptor *fildes*. File descriptor flags are associated with a single file
 10786 descriptor and do not affect other file descriptors that refer to the same file.

10787 **F_SETFD** Set the file descriptor flags defined in <fcntl.h>, that are associated with *fildes*,
 10788 to the third argument, *arg*, taken as type **int**. If the FD_CLOEXEC flag in the
 10789 third argument is 0, the file shall remain open across the *exec* functions;
 10790 otherwise, the file shall be closed upon successful execution of one of the *exec*
 10791 functions.

10792 **F_GETFL** Get the file status flags and file access modes, defined in <fcntl.h>, for the file
 10793 description associated with *fildes*. The file access modes can be extracted from
 10794 the return value using the mask O_ACCMODE, which is defined in <fcntl.h>.
 10795 File status flags and file access modes are associated with the file description
 10796 and do not affect other file descriptors that refer to the same file with different
 10797 open file descriptions.

10798 **F_SETFL** Set the file status flags, defined in <fcntl.h>, for the file description associated
 10799 with *fildes* from the corresponding bits in the third argument, *arg*, taken as
 10800 type **int**. Bits corresponding to the file access mode and the file creation flags,
 10801 as defined in <fcntl.h>, that are set in *arg* shall be ignored. If any bits in *arg*
 10802 other than those mentioned here are changed by the application, the result is
 10803 unspecified.

10804 **F_GETOWN** If *fildes* refers to a socket, get the process or process group ID specified to
 10805 receive SIGURG signals when out-of-band data is available. Positive values
 10806 indicate a process ID; negative values, other than -1, indicate a process group
 10807 ID. If *fildes* does not refer to a socket, the results are unspecified.

10808 **F_SETOWN** If *fildes* refers to a socket, set the process or process group ID specified to
 10809 receive SIGURG signals when out-of-band data is available, using the value of
 10810 the third argument, *arg*, taken as type **int**. Positive values indicate a process
 10811 ID; negative values, other than -1, indicate a process group ID. If *fildes* does
 10812 not refer to a socket, the results are unspecified.

10813	The following values for <i>cmd</i> are available for advisory record locking. Record locking shall be	
10814	supported for regular files, and may be supported for other files.	
10815	F_GETLK	Get the first lock which blocks the lock description pointed to by the third
10816		argument, <i>arg</i> , taken as a pointer to type struct flock , defined in <fcntl.h> .
10817		The information retrieved shall overwrite the information passed to <i>fcntl()</i> in
10818		the structure flock . If no lock is found that would prevent this lock from
10819		being created, then the structure shall be left unchanged except for the lock
10820		type which shall be set to F_UNLCK.
10821	F_SETLK	Set or clear a file segment lock according to the lock description pointed to by
10822		the third argument, <i>arg</i> , taken as a pointer to type struct flock , defined in
10823		<fcntl.h> . F_SETLK can establish shared (or read) locks (F_RDLCK) or
10824		exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock
10825		(F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in <fcntl.h> .
10826		If a shared or exclusive lock cannot be set, <i>fcntl()</i> shall return immediately
10827		with a return value of -1.
10828	F_SETLKW	This command shall be equivalent to F_SETLK except that if a shared or
10829		exclusive lock is blocked by other locks, the thread shall wait until the request
10830		can be satisfied. If a signal that is to be caught is received while <i>fcntl()</i> is
10831		waiting for a region, <i>fcntl()</i> shall be interrupted. Upon return from the signal
10832		handler, <i>fcntl()</i> shall return -1 with <i>errno</i> set to [EINTR], and the lock
10833		operation shall not be done.
10834	Additional implementation-defined values for <i>cmd</i> may be defined in <fcntl.h> . Their names	
10835	shall start with F_.	
10836	When a shared lock is set on a segment of a file, other processes shall be able to set shared locks	
10837	on that segment or a portion of it. A shared lock prevents any other process from setting an	
10838	exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the	
10839	file descriptor was not opened with read access.	
10840	An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock	
10841	on any portion of the protected area. A request for an exclusive lock shall fail if the file	
10842	descriptor was not opened with write access.	
10843	The structure flock describes the type (<i>l_type</i>), starting offset (<i>l_whence</i>), relative offset (<i>l_start</i>),	
10844	size (<i>l_len</i>), and process ID (<i>l_pid</i>) of the segment of the file to be affected.	
10845	The value of <i>l_whence</i> is SEEK_SET, SEEK_CUR, or SEEK_END, to indicate that the relative	
10846	offset <i>l_start</i> bytes shall be measured from the start of the file, current position, or end of the file,	
10847	respectively. The value of <i>l_len</i> is the number of consecutive bytes to be locked. The value of	
10848	<i>l_len</i> may be negative (where the definition of off_t permits negative values of <i>l_len</i>). The <i>l_pid</i>	
10849	field is only used with F_GETLK to return the process ID of the process holding a blocking lock.	
10850	After a successful F_GETLK request, when a blocking lock is found, the values returned in the	
10851	flock structure shall be as follows:	
10852	<i>l_type</i>	Type of blocking lock found.
10853	<i>l_whence</i>	SEEK_SET.
10854	<i>l_start</i>	Start of the blocking lock.
10855	<i>l_len</i>	Length of the blocking lock.
10856	<i>l_pid</i>	Process ID of the process that holds the blocking lock.

10857 If the command is F_SETLKW and the process must wait for another process to release a lock,
 10858 then the range of bytes to be locked shall be determined before the *fcntl()* function blocks. If the
 10859 file size or file descriptor seek offset change while *fcntl()* is blocked, this shall not affect the
 10860 range of bytes locked.

10861 If *l_len* is positive, the area affected shall start at *l_start* and end at *l_start+l_len-1*. If *l_len* is
 10862 negative, the area affected shall start at *l_start+l_len* and end at *l_start-1*. Locks may start and
 10863 extend beyond the current end of a file, but shall not extend before the beginning of the file. A
 10864 lock shall be set to extend to the largest possible value of the file offset for that file by setting
 10865 *l_len* to 0. If such a lock also has *l_start* set to 0 and *l_whence* is set to SEEK_SET, the whole file
 10866 shall be locked.

10867 There shall be at most one type of lock set for each byte in the file. Before a successful return
 10868 from an F_SETLK or an F_SETLKW request when the calling process has previously existing
 10869 locks on bytes in the region specified by the request, the previous lock type for each byte in the
 10870 specified region shall be replaced by the new lock type. As specified above under the
 10871 descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request
 10872 (respectively) shall fail or block when another process has existing locks on bytes in the specified
 10873 region and the type of any of those locks conflicts with the type specified in the request.

10874 All locks associated with a file for a given process shall be removed when a file descriptor for
 10875 that file is closed by that process or the process holding that file descriptor terminates. Locks are
 10876 not inherited by a child process.

10877 A potential for deadlock occurs if a process controlling a locked region is put to sleep by
 10878 attempting to lock another process' locked region. If the system detects that sleeping until a
 10879 locked region is unlocked would cause a deadlock, *fcntl()* shall fail with an [EDEADLK] error.

10880 An unlock (F_UNLCK) request in which *l_len* is non-zero and the offset of the last byte of the
 10881 requested segment is the maximum value for an object of type **off_t**, when the process has an
 10882 existing lock in which *l_len* is 0 and which includes the last byte of the requested segment, shall
 10883 be treated as a request to unlock from the start of the requested segment with an *l_len* equal to 0.
 10884 Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.

10885 SHM When the file descriptor *filides* refers to a shared memory object, the behavior of *fcntl()* shall be
 10886 the same as for a regular file except the effect of the following values for the argument *cmd* shall
 10887 be unspecified: F_SETFL, F_GETLK, F_SETLK, and F_SETLKW.

10888 TYM If *filides* refers to a typed memory object, the result of the *fcntl()* function is unspecified.

10889 RETURN VALUE

10890 Upon successful completion, the value returned shall depend on *cmd* as follows:

10891	F_DUPFD	A new file descriptor.
10892	F_GETFD	Value of flags defined in <fcntl.h>. The return value shall not be negative.
10893	F_SETFD	Value other than -1.
10894	F_GETFL	Value of file status flags and access modes. The return value is not negative.
10895	F_SETFL	Value other than -1.
10896	F_GETLK	Value other than -1.
10897	F_SETLK	Value other than -1.
10898	F_SETLKW	Value other than -1.
10899	F_GETOWN	Value of the socket owner process or process group; this will not be -1.

10900	F_SETOWN	Value other than -1.
10901		Otherwise, -1 shall be returned and <i>errno</i> set to indicate the error.
10902	ERRORS	
10903		The <i>fcntl()</i> function shall fail if:
10904	[EACCES] or [EAGAIN]	
10905		The <i>cmd</i> argument is F_SETLK; the type of lock (<i>l_type</i>) is a shared (F_RDLCK) or exclusive (F_WRLCK) lock and the segment of a file to be locked is already exclusive-locked by another process, or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.
10910	[EBADF]	The <i>fildest</i> argument is not a valid open file descriptor, or the argument <i>cmd</i> is F_SETLK or F_SETLKW, the type of lock, <i>l_type</i> , is a shared lock (F_RDLCK), and <i>fildest</i> is not a valid file descriptor open for reading, or the type of lock, <i>l_type</i> , is an exclusive lock (F_WRLCK), and <i>fildest</i> is not a valid file descriptor open for writing.
10915	[EINTR]	The <i>cmd</i> argument is F_SETLKW and the function was interrupted by a signal.
10916	[EINVAL]	The <i>cmd</i> argument is invalid, or the <i>cmd</i> argument is F_DUPFD and <i>arg</i> is negative or greater than or equal to {OPEN_MAX}, or the <i>cmd</i> argument is F_GETLK, F_SETLK, or F_SETLKW and the data pointed to by <i>arg</i> is not valid, or <i>fildest</i> refers to a file that does not support locking.
10920	[EMFILE]	The argument <i>cmd</i> is F_DUPFD and {OPEN_MAX} file descriptors are currently open in the calling process, or no file descriptors greater than or equal to <i>arg</i> are available.
10923	[ENOLCK]	The argument <i>cmd</i> is F_SETLK or F_SETLKW and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.
10926	[EOVERFLOW]	One of the values to be returned cannot be represented correctly.
10927	[EOVERFLOW]	The <i>cmd</i> argument is F_GETLK, F_SETLK, or F_SETLKW and the smallest or, if <i>l_len</i> is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type off_t .
10930		The <i>fcntl()</i> function may fail if:
10931	[EDEADLK]	The <i>cmd</i> argument is F_SETLKW, the lock is blocked by a lock from another process, and putting the calling process to sleep to wait for that lock to become free would cause a deadlock.

EXAMPLES

10935 None.

APPLICATION USAGE

10937 None.

RATIONALE

10939 The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number of arguments. It is used because System V uses pointers for the implementation of file locking functions.

10942 The *arg* values to F_GETFD, F_SETFD, F_GETFL, and F_SETFL all represent flag values to allow for future growth. Applications using these functions should do a read-modify-write operation

10944 on them, rather than assuming that only the values defined by this volume of
10945 IEEE Std 1003.1-2001 are valid. It is a common error to forget this, particularly in the case of
10946 F_SETFD.

10947 This volume of IEEE Std 1003.1-2001 permits concurrent read and write access to file data using
10948 the *fcntl()* function; this is a change from the 1984 /usr/group standard and early proposals.
10949 Without concurrency controls, this feature may not be fully utilized without occasional loss of
10950 data.

10951 Data losses occur in several ways. One case occurs when several processes try to update the
10952 same record, without sequencing controls; several updates may occur in parallel and the last
10953 writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing
10954 reorganization. Without exclusive use to the tree segment by the updating process, other reading
10955 processes chance getting lost in the database when the index blocks are split, condensed,
10956 inserted, or deleted. While *fcntl()* is useful for many applications, it is not intended to be overly
10957 general and does not handle the bit-tree example well.

10958 This facility is only required for regular files because it is not appropriate for many devices such
10959 as terminals and network connections.

10960 Since *fcntl()* works with “any file descriptor associated with that file, however it is obtained”,
10961 the file descriptor may have been inherited through a *fork()* or *exec* operation and thus may
10962 affect a file that another process also has open.

10963 The use of the open file description to identify what to lock requires extra calls and presents
10964 problems if several processes are sharing an open file description, but there are too many
10965 implementations of the existing mechanism for this volume of IEEE Std 1003.1-2001 to use
10966 different specifications.

10967 Another consequence of this model is that closing any file descriptor for a given file (whether or
10968 not it is the same open file description that created the lock) causes the locks on that file to be
10969 relinquished for that process. Equivalently, any close for any file/process pair relinquishes the
10970 locks owned on that file for that process. But note that while an open file description may be
10971 shared through *fork()*, locks are not inherited through *fork()*. Yet locks may be inherited through
10972 one of the *exec* functions.

10973 The identification of a machine in a network environment is outside the scope of this volume of
10974 IEEE Std 1003.1-2001. Thus, an *l_sysid* member, such as found in System V, is not included in the
10975 locking structure.

10976 Changing of lock types can result in a previously locked region being split into smaller regions.

10977 Mandatory locking was a major feature of the 1984 /usr/group standard.

10978 For advisory file record locking to be effective, all processes that have access to a file must
10979 cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode
10980 record locking is important when it cannot be assumed that all processes are cooperating. For
10981 example, if one user uses an editor to update a file at the same time that a second user executes
10982 another process that updates the same file and if only one of the two processes is using advisory
10983 locking, the processes are not cooperating. Enforcement-mode record locking would protect
10984 against accidental collisions.

10985 Secondly, advisory record locking requires a process using locking to bracket each I/O operation
10986 with lock (or test) and unlock operations. With enforcement-mode file and record locking, a
10987 process can lock the file once and unlock when all I/O operations have been completed.
10988 Enforcement-mode record locking provides a base that can be enhanced; for example, with
10989 sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so
10990 other processes could read it, but none of them could write it.

- 10991 Mandatory locks were omitted for several reasons:
- 10992 1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most
10993 implementations; this was confusing, at best.
 - 10994 2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
 - 10995 3. Any publicly readable file could be locked by anyone. Many historical implementations
10996 keep the password database in a publicly readable file. A malicious user could thus
10997 prohibit logins. Another possibility would be to hold open a long-distance telephone line.
 - 10998 4. Some demand-paged historical implementations offer memory mapped files, and
10999 enforcement cannot be done on that type of file.
- 11000 Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a
11001 timeout facility in applications requiring it. This is useful in deadlock detection. Since
11002 implementation of full deadlock detection is not always feasible, the [EDEADLK] error was
11003 made optional.
- 11004 **FUTURE DIRECTIONS**
- 11005 None.
- 11006 **SEE ALSO**
- 11007 *alarm()*, *close()*, *exec*, *open()*, *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11008 **<fcntl.h>**, **<signal.h>**, **<unistd.h>**
- 11009 **CHANGE HISTORY**
- 11010 First released in Issue 1. Derived from Issue 1 of the SVID.
- 11011 **Issue 5**
- 11012 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
11013 Threads Extension.
- 11014 Large File Summit extensions are added.
- 11015 **Issue 6**
- 11016 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.
- 11017 The following new requirements on POSIX implementations derive from alignment with the
11018 Single UNIX Specification:
- 11019 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
11020 required for conforming implementations of previous POSIX specifications, it was not
11021 required for UNIX applications.
 - 11022 • In the DESCRIPTION, sentences describing behavior when *l_len* is negative are now
11023 mandated, and the description of unlock (F_UNLOCK) when *l_len* is non-negative is
11024 mandated.
 - 11025 • In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd* is
11026 invalid, and two [EOVERFLOW] error conditions are added.
- 11027 The F_GETOWN and F_SETOWN values are added for sockets.
- 11028 The following changes were made to align with the IEEE P1003.1a draft standard:
- 11029 • Clarification is added that the extent of the bytes locked is determined prior to the blocking
11030 action.
- 11031 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
11032 *fcntl()* results are unspecified for typed memory objects.

11033

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

11034 **NAME**11035 fcvt — convert a floating-point number to a string (**LEGACY**)11036 **SYNOPSIS**

11037 xSI #include <stdlib.h>

11038 char *fcvt(double *value*, int *ndigit*, int *restrict *decpt*,
11039 int *restrict *sign*);

11040

11041 **DESCRIPTION**11042 Refer to *ecvt()*.

11043 NAME

11044 `fdatasync` — synchronize the data of a file (**REALTIME**)

11045 SYNOPSIS

11046 `SIO` `#include <unistd.h>`

11047 `int fdatasync(int fildes);`

11048

11049 DESCRIPTION

11050 The `fdatasync()` function shall force all currently queued I/O operations associated with the file
11051 indicated by file descriptor *fil*des to the synchronized I/O completion state.

11052 The functionality shall be equivalent to `fsync()` with the symbol `_POSIX_SYNCHRONIZED_IO`
11053 defined, with the exception that all I/O operations shall be completed as defined for
11054 synchronized I/O data integrity completion.

11055 RETURN VALUE

11056 If successful, the `fdatasync()` function shall return the value 0; otherwise, the function shall return
11057 the value `-1` and set *errno* to indicate the error. If the `fdatasync()` function fails, outstanding I/O
11058 operations are not guaranteed to have been completed.

11059 ERRORS

11060 The `fdatasync()` function shall fail if:

11061 [EBADF] The *fil*des argument is not a valid file descriptor open for writing.

11062 [EINVAL] This implementation does not support synchronized I/O for this file.

11063 In the event that any of the queued I/O operations fail, `fdatasync()` shall return the error
11064 conditions defined for `read()` and `write()`.

11065 EXAMPLES

11066 None.

11067 APPLICATION USAGE

11068 None.

11069 RATIONALE

11070 None.

11071 FUTURE DIRECTIONS

11072 None.

11073 SEE ALSO

11074 `aio_fsync()`, `fcntl()`, `fsync()`, `open()`, `read()`, `write()`, the Base Definitions volume of
11075 IEEE Std 1003.1-2001, `<unistd.h>`

11076 CHANGE HISTORY

11077 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11078 Issue 6

11079 The [ENOSYS] error condition has been removed as stubs need not be provided if an
11080 implementation does not support the Synchronized Input and Output option.

11081 The `fdatasync()` function is marked as part of the Synchronized Input and Output option.

11082 **NAME**11083 fdetach — detach a name from a STREAMS-based file descriptor (**STREAMS**)11084 **SYNOPSIS**

11085 xSR #include <stropts.h>

11086 int fdetach(const char *path);

11087

11088 **DESCRIPTION**

11089 The *fdetach()* function shall detach a STREAMS-based file from the file to which it was attached
 11090 by a previous call to *fattach()*. The *path* argument points to the pathname of the attached
 11091 STREAMS file. The process shall have appropriate privileges or be the owner of the file. A
 11092 successful call to *fdetach()* shall cause all pathnames that named the attached STREAMS file to
 11093 again name the file to which the STREAMS file was attached. All subsequent operations on *path*
 11094 shall operate on the underlying file and not on the STREAMS file.

11095 All open file descriptions established while the STREAMS file was attached to the file referenced
 11096 by *path* shall still refer to the STREAMS file after the *fdetach()* has taken effect.

11097 If there are no open file descriptors or other references to the STREAMS file, then a successful
 11098 call to *fdetach()* shall be equivalent to performing the last *close()* on the attached file.

11099 **RETURN VALUE**

11100 Upon successful completion, *fdetach()* shall return 0; otherwise, it shall return -1 and set *errno* to
 11101 indicate the error.

11102 **ERRORS**11103 The *fdetach()* function shall fail if:

11104 [EACCES] Search permission is denied on a component of the path prefix.

11105 [EINVAL] The *path* argument names a file that is not currently attached.

11106 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 11107 argument.

11108 [ENAMETOOLONG]

11109 The size of a pathname exceeds {PATH_MAX} or a pathname component is
 11110 longer than {NAME_MAX}.

11111 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

11112 [ENOTDIR] A component of the path prefix is not a directory.

11113 [EPERM] The effective user ID is not the owner of *path* and the process does not have
 11114 appropriate privileges.

11115 The *fdetach()* function may fail if:

11116 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 11117 resolution of the *path* argument.

11118 [ENAMETOOLONG]

11119 Pathname resolution of a symbolic link produced an intermediate result
 11120 whose length exceeds {PATH_MAX}.

11121 **EXAMPLES**11122 **Detaching a File**

11123 The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to
11124 which it was attached by a previous, successful call to *fattach()*. Subsequent calls to open this
11125 file refer to the underlying file, not to the STREAMS file.

```
11126 #include <stropts.h>
11127 ...
11128     char *filename = "/tmp/named-STREAM";
11129     int ret;
11130     ret = fdetach(filename);
```

11131 **APPLICATION USAGE**

11132 None.

11133 **RATIONALE**

11134 None.

11135 **FUTURE DIRECTIONS**

11136 None.

11137 **SEE ALSO**

11138 *fattach()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stropts.h>

11139 **CHANGE HISTORY**

11140 First released in Issue 4, Version 2.

11141 **Issue 5**

11142 Moved from X/OPEN UNIX extension to BASE.

11143 **Issue 6**

11144 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

11145 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
11146 [ELOOP] error condition is added.

11147 **NAME**

11148 fdim, fdimf, fdiml — compute positive difference between two floating-point numbers

11149 **SYNOPSIS**

11150 #include <math.h>

11151 double fdim(double x, double y);

11152 float fdimf(float x, float y);

11153 long double fdiml(long double x, long double y);

11154 **DESCRIPTION**

11155 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11156 conflict between the requirements described here and the ISO C standard is unintentional. This
 11157 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11158 These functions shall determine the positive difference between their arguments. If *x* is greater
 11159 than *y*, *x*−*y* is returned. If *x* is less than or equal to *y*, +0 is returned.

11160 An application wishing to check for error situations should set *errno* to zero and call
 11161 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 11162 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 11163 zero, an error has occurred.

11164 **RETURN VALUE**

11165 Upon successful completion, these functions shall return the positive difference value.

11166 If *x*−*y* is positive and overflows, a range error shall occur and *fdim*(), *fdimf*(), and *fdiml*() shall
 11167 return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

11168 XSI If *x*−*y* is positive and underflows, a range error may occur, and either (*x*−*y*) (if representable), or
 11169 0.0 (if supported), or an implementation-defined value shall be returned.

11170 MX If *x* or *y* is NaN, a NaN shall be returned.

11171 **ERRORS**

11172 The *fdim*() function shall fail if:

11173 Range Error The result overflows.

11174 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 11175 then *errno* shall be set to [ERANGE]. If the integer expression
 11176 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 11177 floating-point exception shall be raised.

11178 The *fdim*() function may fail if:

11179 Range Error The result underflows.

11180 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 11181 then *errno* shall be set to [ERANGE]. If the integer expression
 11182 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 11183 floating-point exception shall be raised.

11184 EXAMPLES

11185 None.

11186 APPLICATION USAGE

11187 On implementations supporting IEEE Std 754-1985, $x-y$ cannot underflow, and hence the 0.0
11188 return value is shaded as an extension for implementations supporting the XSI extension rather
11189 than an MX extension.

11190 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
11191 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

11192 RATIONALE

11193 None.

11194 FUTURE DIRECTIONS

11195 None.

11196 SEE ALSO

11197 *feclearexcept()*, *fetestexcept()*, *fmax()*, *fmin()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11198 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

11199 CHANGE HISTORY

11200 First released in Issue 6. Derived from the ISO / IEC 9899:1999 standard.

11201 **NAME**

11202 fdopen — associate a stream with a file descriptor

11203 **SYNOPSIS**11204 CX `#include <stdio.h>`11205 `FILE *fdopen(int fil-des, const char *mode);`

11206

11207 **DESCRIPTION**11208 The *fdopen()* function shall associate a stream with a file descriptor.11209 The *mode* argument is a character string having one of the following values:11210 *r* or *rb* Open a file for reading.11211 *w* or *wb* Open a file for writing.11212 *a* or *ab* Open a file for writing at end-of-file.11213 *r+* or *rb+* or *r+b* Open a file for update (reading and writing).11214 *w+* or *wb+* or *w+b* Open a file for update (reading and writing).11215 *a+* or *ab+* or *a+b* Open a file for update (reading and writing) at end-of-file.11216 The meaning of these flags is exactly as specified in *fopen()*, except that modes beginning with *w*
11217 shall not cause truncation of the file.11218 Additional values for the *mode* argument may be supported by an implementation.11219 The application shall ensure that the mode of the stream as expressed by the *mode* argument is
11220 allowed by the file access mode of the open file description to which *fil-des* refers. The file
11221 position indicator associated with the new stream is set to the position indicated by the file
11222 offset associated with the file descriptor.11223 The error and end-of-file indicators for the stream shall be cleared. The *fdopen()* function may
11224 cause the *st_atime* field of the underlying file to be marked for update.11225 SHM If *fil-des* refers to a shared memory object, the result of the *fdopen()* function is unspecified.11226 TYM If *fil-des* refers to a typed memory object, the result of the *fdopen()* function is unspecified.11227 The *fdopen()* function shall preserve the offset maximum previously set for the open file
11228 description corresponding to *fil-des*.11229 **RETURN VALUE**11230 Upon successful completion, *fdopen()* shall return a pointer to a stream; otherwise, a null pointer
11231 shall be returned and *errno* set to indicate the error.11232 **ERRORS**11233 The *fdopen()* function may fail if:11234 [EBADF] The *fil-des* argument is not a valid file descriptor.11235 [EINVAL] The *mode* argument is not a valid mode.

11236 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

11237 [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

11238 [ENOMEM] Insufficient space to allocate a buffer.

11239 EXAMPLES

11240 None.

11241 APPLICATION USAGE

11242 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but
 11243 do not return streams.

11244 RATIONALE

11245 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, or *fcntl()*;
 11246 inherited through *fork()* or *exec*; or perhaps obtained by implementation-defined means, such as
 11247 the 4.3 BSD *socket()* call.

11248 The meanings of the *mode* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, open for write
 11249 (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. The *mode* argument
 11250 formats that include *a b* are allowed for consistency with the ISO C standard function *fopen()*.
 11251 The *b* has no effect on the resulting stream. Although not explicitly required by this volume of
 11252 IEEE Std 1003.1-2001, a good implementation of append (*a*) mode would cause the *O_APPEND*
 11253 flag to be set.

11254 FUTURE DIRECTIONS

11255 None.

11256 SEE ALSO

11257 Section 2.5.1 (on page 35), *fclose()*, *fopen()*, *open()*, the Base Definitions volume of
 11258 IEEE Std 1003.1-2001, **<stdio.h>**

11259 CHANGE HISTORY

11260 First released in Issue 1. Derived from Issue 1 of the SVID.

11261 Issue 5

11262 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

11263 Large File Summit extensions are added.

11264 Issue 6

11265 The following new requirements on POSIX implementations derive from alignment with the
 11266 Single UNIX Specification:

- 11267 • In the DESCRIPTION, the use and setting of the *mode* argument are changed to include
 11268 binary streams.
- 11269 • In the DESCRIPTION, text is added for large file support to indicate setting of the offset
 11270 maximum in the open file description.
- 11271 • All errors identified in the ERRORS section are added.
- 11272 • In the DESCRIPTION, text is added that the *fdopen()* function may cause *st_atime* to be
 11273 updated.

11274 The following changes were made to align with the IEEE P1003.1a draft standard:

- 11275 • Clarification is added that it is the responsibility of the application to ensure that the mode is
 11276 compatible with the open file descriptor.

11277 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 11278 *fdopen()* results are unspecified for typed memory objects.

11279 **NAME**

11280 feclearexcept — clear floating-point exception

11281 **SYNOPSIS**

11282 #include <fenv.h>

11283 int feclearexcept(int *excepts*);11284 **DESCRIPTION**

11285 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11286 conflict between the requirements described here and the ISO C standard is unintentional. This
11287 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11288 The *feclearexcept()* function shall attempt to clear the supported floating-point exceptions
11289 represented by *excepts*.

11290 **RETURN VALUE**

11291 If the argument is zero or if all the specified exceptions were successfully cleared, *feclearexcept()*
11292 shall return zero. Otherwise, it shall return a non-zero value.

11293 **ERRORS**

11294 No errors are defined.

11295 **EXAMPLES**

11296 None.

11297 **APPLICATION USAGE**

11298 None.

11299 **RATIONALE**

11300 None.

11301 **FUTURE DIRECTIONS**

11302 None.

11303 **SEE ALSO**

11304 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fetestexcept()*, the Base Definitions volume of
11305 IEEE Std 1003.1-2001, <**fenv.h**>

11306 **CHANGE HISTORY**

11307 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11308 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11309 **NAME**

11310 fegetenv, fesetenv — get and set current floating-point environment

11311 **SYNOPSIS**

11312 #include <fenv.h>

11313 int fegetenv(fenv_t *envp);

11314 int fesetenv(const fenv_t *envp);

11315 **DESCRIPTION**

11316 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 11317 conflict between the requirements described here and the ISO C standard is unintentional. This
 11318 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11319 The *fegetenv()* function shall attempt to store the current floating-point environment in the object
 11320 pointed to by *envp*.

11321 The *fesetenv()* function shall attempt to establish the floating-point environment represented by
 11322 the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to
 11323 *fegetenv()* or *feholdexcept()*, or equal a floating-point environment macro. The *fesetenv()* function
 11324 does not raise floating-point exceptions, but only installs the state of the floating-point status
 11325 flags represented through its argument.

11326 **RETURN VALUE**

11327 If the representation was successfully stored, *fegetenv()* shall return zero. Otherwise, it shall
 11328 return a non-zero value. If the environment was successfully established, *fesetenv()* shall return
 11329 zero. Otherwise, it shall return a non-zero value.

11330 **ERRORS**

11331 No errors are defined.

11332 **EXAMPLES**

11333 None.

11334 **APPLICATION USAGE**

11335 None.

11336 **RATIONALE**

11337 None.

11338 **FUTURE DIRECTIONS**

11339 None.

11340 **SEE ALSO**11341 *feholdexcept()*, *feupdateenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <fenv.h>11342 **CHANGE HISTORY**

11343 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11344 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11345 **NAME**

11346 fegetexceptflag, fesetexceptflag — get and set floating-point status flags

11347 **SYNOPSIS**

11348 #include <fenv.h>

11349 int fegetexceptflag(fexcept_t *flagp, int excepts);

11350 int fesetexceptflag(const fexcept_t *flagp, int excepts);

11351 **DESCRIPTION**

11352 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 11353 conflict between the requirements described here and the ISO C standard is unintentional. This
 11354 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11355 The *fegetexceptflag()* function shall attempt to store an implementation-defined representation of
 11356 the states of the floating-point status flags indicated by the argument *excepts* in the object
 11357 pointed to by the argument *flagp*.

11358 The *fesetexceptflag()* function shall attempt to set the floating-point status flags indicated by the
 11359 argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by
 11360 *flagp* shall have been set by a previous call to *fegetexceptflag()* whose second argument
 11361 represented at least those floating-point exceptions represented by the argument *excepts*. This
 11362 function does not raise floating-point exceptions, but only sets the state of the flags.

11363 **RETURN VALUE**

11364 If the representation was successfully stored, *fegetexceptflag()* shall return zero. Otherwise, it
 11365 shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions
 11366 were successfully set, *fesetexceptflag()* shall return zero. Otherwise, it shall return a non-zero
 11367 value.

11368 **ERRORS**

11369 No errors are defined.

11370 **EXAMPLES**

11371 None.

11372 **APPLICATION USAGE**

11373 None.

11374 **RATIONALE**

11375 None.

11376 **FUTURE DIRECTIONS**

11377 None.

11378 **SEE ALSO**

11379 *feclearexcept()*, *feraiseexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 11380 <fenv.h>

11381 **CHANGE HISTORY**

11382 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11383 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11384 NAME

11385 fegetround, fesetround — get and set current rounding direction

11386 SYNOPSIS

```
11387       #include <fenv.h>

11388       int fegetround(void);
11389       int fesetround(int round);
```

11390 DESCRIPTION

11391 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 11392 conflict between the requirements described here and the ISO C standard is unintentional. This
 11393 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11394 The *fegetround()* function shall get the current rounding direction.

11395 The *fesetround()* function shall establish the rounding direction represented by its argument
 11396 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding
 11397 direction is not changed.

11398 RETURN VALUE

11399 The *fegetround()* function shall return the value of the rounding direction macro representing the
 11400 current rounding direction or a negative value if there is no such rounding direction macro or
 11401 the current rounding direction is not determinable.

11402 The *fesetround()* function shall return a zero value if and only if the requested rounding direction
 11403 was established.

11404 ERRORS

11405 No errors are defined.

11406 EXAMPLES

11407 The following example saves, sets, and restores the rounding direction, reporting an error and
 11408 aborting if setting the rounding direction fails:

```
11409       #include <fenv.h>
11410       #include <assert.h>
11411       void f(int round_dir)
11412       {
11413           #pragma STDC FENV_ACCESS ON
11414           int save_round;
11415           int setround_ok;
11416           save_round = fegetround();
11417           setround_ok = fesetround(round_dir);
11418           assert(setround_ok == 0);
11419           /* ... */
11420           fesetround(save_round);
11421           /* ... */
11422       }
```

11423 APPLICATION USAGE

11424 None.

11425 RATIONALE

11426 None.

11427 **FUTURE DIRECTIONS**

11428 None.

11429 **SEE ALSO**

11430 The Base Definitions volume of IEEE Std 1003.1-2001, <fenv.h>

11431 **CHANGE HISTORY**

11432 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11433 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11434 **NAME**

11435 feholdexcept — save current floating-point environment

11436 **SYNOPSIS**

11437 #include <fenv.h>

11438 int feholdexcept(fenv_t *envp);

11439 **DESCRIPTION**

11440 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11441 conflict between the requirements described here and the ISO C standard is unintentional. This
11442 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11443 The *feholdexcept()* function shall save the current floating-point environment in the object
11444 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on
11445 floating-point exceptions) mode, if available, for all floating-point exceptions.

11446 **RETURN VALUE**

11447 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception
11448 handling was successfully installed.

11449 **ERRORS**

11450 No errors are defined.

11451 **EXAMPLES**

11452 None.

11453 **APPLICATION USAGE**

11454 None.

11455 **RATIONALE**

11456 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard
11457 implementations which have the default non-stop mode and at least one other mode for trap
11458 handling or aborting. If the implementation provides only the non-stop mode, then installing the
11459 non-stop mode is trivial.

11460 **FUTURE DIRECTIONS**

11461 None.

11462 **SEE ALSO**

11463 *fegetenv()*, *fesetenv()*, *feupdateenv()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11464 <fenv.h>

11465 **CHANGE HISTORY**

11466 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11467 NAME

11468 feof — test end-of-file indicator on a stream

11469 SYNOPSIS

11470 #include <stdio.h>

11471 int feof(FILE *stream);

11472 DESCRIPTION

11473 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11474 conflict between the requirements described here and the ISO C standard is unintentional. This
11475 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11476 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.

11477 RETURN VALUE

11478 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.

11479 ERRORS

11480 No errors are defined.

11481 EXAMPLES

11482 None.

11483 APPLICATION USAGE

11484 None.

11485 RATIONALE

11486 None.

11487 FUTURE DIRECTIONS

11488 None.

11489 SEE ALSO

11490 *clearerr()*, *ferror()*, *fopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

11491 CHANGE HISTORY

11492 First released in Issue 1. Derived from Issue 1 of the SVID.

11493 **NAME**11494 `feraiseexcept` — raise floating-point exception11495 **SYNOPSIS**11496 `#include <fenv.h>`11497 `int fraiseexcept(int excepts);`11498 **DESCRIPTION**

11499 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 11500 conflict between the requirements described here and the ISO C standard is unintentional. This
 11501 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11502 The `feraiseexcept()` function shall attempt to raise the supported floating-point exceptions
 11503 represented by the argument *excepts*. The order in which these floating-point exceptions are
 11504 raised is unspecified. Whether the `feraiseexcept()` function additionally raises the inexact
 11505 floating-point exception whenever it raises the overflow or underflow floating-point exception is
 11506 implementation-defined.

11507 **RETURN VALUE**

11508 If the argument is zero or if all the specified exceptions were successfully raised, `feraiseexcept()`
 11509 shall return zero. Otherwise, it shall return a non-zero value.

11510 **ERRORS**

11511 No errors are defined.

11512 **EXAMPLES**

11513 None.

11514 **APPLICATION USAGE**

11515 The effect is intended to be similar to that of floating-point exceptions raised by arithmetic
 11516 operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

11517 **RATIONALE**

11518 Raising overflow or underflow is allowed to also raise inexact because on some architectures the
 11519 only practical way to raise an exception is to execute an instruction that has the exception as a
 11520 side effect. The function is not restricted to accept only valid coincident expressions for atomic
 11521 operations, so the function can be used to raise exceptions accrued over several operations.

11522 **FUTURE DIRECTIONS**

11523 None.

11524 **SEE ALSO**

11525 `feclearexcept()`, `fegetexceptflag()`, `fesetexceptflag()`, `fetestexcept()`, the Base Definitions volume of
 11526 IEEE Std 1003.1-2001, **<fenv.h>**

11527 **CHANGE HISTORY**

11528 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11529 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11530 **NAME**11531 *ferror* — test error indicator on a stream11532 **SYNOPSIS**11533 `#include <stdio.h>`11534 `int ferror(FILE *stream);`11535 **DESCRIPTION**

11536 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11537 conflict between the requirements described here and the ISO C standard is unintentional. This
11538 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11539 The *ferror()* function shall test the error indicator for the stream pointed to by *stream*.11540 **RETURN VALUE**11541 The *ferror()* function shall return non-zero if and only if the error indicator is set for *stream*.11542 **ERRORS**

11543 No errors are defined.

11544 **EXAMPLES**

11545 None.

11546 **APPLICATION USAGE**

11547 None.

11548 **RATIONALE**

11549 None.

11550 **FUTURE DIRECTIONS**

11551 None.

11552 **SEE ALSO**11553 *clearerr()*, *feof()*, *fopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**11554 **CHANGE HISTORY**

11555 First released in Issue 1. Derived from Issue 1 of the SVID.

11556 **NAME**

11557 fesetenv — set current floating-point environment

11558 **SYNOPSIS**

11559 #include <fenv.h>

11560 int fesetenv(const fenv_t *envp);

11561 **DESCRIPTION**11562 Refer to *fegetenv()*.

11563 **NAME**

11564 fesetexceptflag — set floating-point status flags

11565 **SYNOPSIS**

11566 #include <fenv.h>

11567 int fesetexceptflag(const fexcept_t *flagp, int excepts);

11568 **DESCRIPTION**

11569 Refer to *fegetexceptflag()*.

11570 **NAME**

11571 fesetround — set current rounding direction

11572 **SYNOPSIS**

11573 #include <fenv.h>

11574 int fesetround(int *round*) ;

11575 **DESCRIPTION**

11576 Refer to *fegetround()*.

11577 **NAME**

11578 fetestexcept — test floating-point exception flags

11579 **SYNOPSIS**

11580 #include <fenv.h>

11581 int fetestexcept(int *excepts*);11582 **DESCRIPTION**

11583 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 11584 conflict between the requirements described here and the ISO C standard is unintentional. This
 11585 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11586 The *fetestexcept()* function shall determine which of a specified subset of the floating-point
 11587 exception flags are currently set. The *excepts* argument specifies the floating-point status flags to
 11588 be queried.

11589 **RETURN VALUE**

11590 The *fetestexcept()* function shall return the value of the bitwise-inclusive OR of the floating-point
 11591 exception macros corresponding to the currently set floating-point exceptions included in
 11592 *excepts*.

11593 **ERRORS**

11594 No errors are defined.

11595 **EXAMPLES**

11596 The following example calls function *f()* if an invalid exception is set, and then function *g()* if an
 11597 overflow exception is set:

```

11598       #include <fenv.h>
11599       /* ... */
11600       {
11601           #pragma STDC FENV_ACCESS ON
11602           int set_excepts;
11603           feclearexcept(FE_INVALID | FE_OVERFLOW);
11604           // maybe raise exceptions
11605           set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
11606           if (set_excepts & FE_INVALID) f();
11607           if (set_excepts & FE_OVERFLOW) g();
11608           /* ... */
11609       }
```

11610 **APPLICATION USAGE**

11611 None.

11612 **RATIONALE**

11613 None.

11614 **FUTURE DIRECTIONS**

11615 None.

11616 **SEE ALSO**

11617 *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, the Base Definitions volume of
 11618 IEEE Std 1003.1-2001, <**fenv.h**>

11619 **CHANGE HISTORY**

11620 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.

11621 **NAME**

11622 feupdateenv — update floating-point environment

11623 **SYNOPSIS**

11624 #include <fenv.h>

11625 int feupdateenv(const fenv_t *envp);

11626 **DESCRIPTION**

11627 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 11628 conflict between the requirements described here and the ISO C standard is unintentional. This
 11629 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11630 The *feupdateenv()* function shall attempt to save the currently raised floating-point exceptions in
 11631 its automatic storage, attempt to install the floating-point environment represented by the object
 11632 pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument
 11633 *envp* shall point to an object set by a call to *feholdexcept()* or *fegetenv()*, or equal a floating-point
 11634 environment macro.

11635 **RETURN VALUE**

11636 The *feupdateenv()* function shall return a zero value if and only if all the required actions were
 11637 successfully carried out.

11638 **ERRORS**

11639 No errors are defined.

11640 **EXAMPLES**

11641 The following example shows sample code to hide spurious underflow floating-point
 11642 exceptions:

```
11643 #include <fenv.h>
11644 double f(double x)
11645 {
11646     #pragma STDC FENV_ACCESS ON
11647     double result;
11648     fenv_t save_env;
11649     feholdexcept(&save_env);
11650     // compute result
11651     if (/* test spurious underflow */)
11652         feclearexcept(FE_UNDERFLOW);
11653     feupdateenv(&save_env);
11654     return result;
11655 }
```

11656 **APPLICATION USAGE**

11657 None.

11658 **RATIONALE**

11659 None.

11660 **FUTURE DIRECTIONS**

11661 None.

11662 **SEE ALSO**11663 *fegetenv()*, *feholdexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**fenv.h**>

11664 **CHANGE HISTORY**

- 11665 First released in Issue 6. Derived from the ISO / IEC 9899: 1999 standard.
- 11666 ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

11667 **NAME**

11668 fflush — flush a stream

11669 **SYNOPSIS**

11670 #include <stdio.h>

11671 int fflush(FILE **stream*);11672 **DESCRIPTION**

11673 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11674 conflict between the requirements described here and the ISO C standard is unintentional. This
 11675 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11676 If *stream* points to an output stream or an update stream in which the most recent operation was
 11677 CX not input, *fflush()* shall cause any unwritten data for that stream to be written to the file, and the
 11678 *st_ctime* and *st_mtime* fields of the underlying file shall be marked for update.

11679 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the
 11680 behavior is defined above.

11681 **RETURN VALUE**

11682 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for
 11683 CX the stream, return EOF, and set *errno* to indicate the error.

11684 **ERRORS**11685 The *fflush()* function shall fail if:

11686 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 11687 process would be delayed in the write operation.

11688 CX [EBADF] The file descriptor underlying *stream* is not valid.

11689 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

11690 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

11691 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 11692 offset maximum associated with the corresponding stream.

11693 CX [EINTR] The *fflush()* function was interrupted by a signal.

11694 CX [EIO] The process is a member of a background process group attempting to write
 11695 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 11696 blocking SIGTTOU, and the process group of the process is orphaned. This
 11697 error may also be returned under implementation-defined conditions.

11698 CX [ENOSPC] There was no free space remaining on the device containing the file.

11699 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 11700 any process. A SIGPIPE signal shall also be sent to the thread.

11701 The *fflush()* function may fail if:

11702 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 11703 capabilities of the device.

11704 **EXAMPLES**11705 **Sending Prompts to Standard Output**

11706 The following example uses *printf()* calls to print a series of prompts for information the user
 11707 must enter from standard input. The *fflush()* calls force the output to standard output. The
 11708 *fflush()* function is used because standard output is usually buffered and the prompt may not
 11709 immediately be printed on the output or terminal. The *gets()* calls read strings from standard
 11710 input and place the results in variables, for use later in the program.

```
11711 #include <stdio.h>
11712 ...
11713 char user[100];
11714 char oldpasswd[100];
11715 char newpasswd[100];
11716 ...
11717 printf("User name: ");
11718 fflush(stdout);
11719 gets(user);

11720 printf("Old password: ");
11721 fflush(stdout);
11722 gets(oldpasswd);

11723 printf("New password: ");
11724 fflush(stdout);
11725 gets(newpasswd);
11726 ...
```

11727 **APPLICATION USAGE**

11728 None.

11729 **RATIONALE**

11730 Data buffered by the system may make determining the validity of the position of the current
 11731 file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()*
 11732 on streams open for *read()* is not mandated by IEEE Std 1003.1-2001.

11733 **FUTURE DIRECTIONS**

11734 None.

11735 **SEE ALSO**

11736 *getrlimit()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

11737 **CHANGE HISTORY**

11738 First released in Issue 1. Derived from Issue 1 of the SVID.

11739 **Issue 5**

11740 Large File Summit extensions are added.

11741 **Issue 6**

11742 Extensions beyond the ISO C standard are marked.

11743 The following new requirements on POSIX implementations derive from alignment with the
 11744 Single UNIX Specification:

- 11745 • The [EFBIG] error is added as part of the large file support extensions.
- 11746 • The [ENXIO] optional error condition is added.

11747 The RETURN VALUE section is updated to note that the error indicator shall be set for the
11748 stream. This is for alignment with the ISO/IEC 9899:1999 standard.

11749 **NAME**

11750 ffs — find first set bit

11751 **SYNOPSIS**

11752 xSI #include <strings.h>

11753 int ffs(int i);

11754

11755 **DESCRIPTION**

11756 The *ffs()* function shall find the first bit set (beginning with the least significant bit) in *i*, and
11757 return the index of that bit. Bits are numbered starting at one (the least significant bit).

11758 **RETURN VALUE**11759 The *ffs()* function shall return the index of the first bit set. If *i* is 0, then *ffs()* shall return 0.11760 **ERRORS**

11761 No errors are defined.

11762 **EXAMPLES**

11763 None.

11764 **APPLICATION USAGE**

11765 None.

11766 **RATIONALE**

11767 None.

11768 **FUTURE DIRECTIONS**

11769 None.

11770 **SEE ALSO**11771 The Base Definitions volume of IEEE Std 1003.1-2001, <**strings.h**>11772 **CHANGE HISTORY**

11773 First released in Issue 4, Version 2.

11774 **Issue 5**

11775 Moved from X/OPEN UNIX extension to BASE.

11776 **NAME**

11777 fgetc — get a byte from a stream

11778 **SYNOPSIS**

11779 #include <stdio.h>

11780 int fgetc(FILE *stream);

11781 **DESCRIPTION**

11782 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11783 conflict between the requirements described here and the ISO C standard is unintentional. This
 11784 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11785 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is
 11786 present, the *fgetc()* function shall obtain the next byte as an **unsigned char** converted to an **int**,
 11787 from the input stream pointed to by *stream*, and advance the associated file position indicator for
 11788 the stream (if defined). Since *fgetc()* operates on bytes, reading a character consisting of multiple
 11789 bytes (or “a multi-byte character”) may require multiple calls to *fgetc()*.

11790 CX The *fgetc()* function may mark the *st_atime* field of the file associated with *stream* for update. The
 11791 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 11792 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 11793 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11794 **RETURN VALUE**

11795 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to
 11796 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the
 11797 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If a read error occurs,
 11798 CX the error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to
 11799 indicate the error.

11800 **ERRORS**11801 The *fgetc()* function shall fail if data needs to be read and:

11802 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 11803 process would be delayed in the *fgetc()* operation.

11804 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 11805 reading.

11806 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 11807 was transferred.

11808 CX [EIO] A physical I/O error has occurred, or the process is in a background process
 11809 group attempting to read from its controlling terminal, and either the process
 11810 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
 11811 This error may also be generated for implementation-defined reasons.

11812 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
 11813 offset maximum associated with the corresponding stream.

11814 The *fgetc()* function may fail if:

11815 CX [ENOMEM] Insufficient storage space is available.

11816 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 11817 capabilities of the device.

11818 EXAMPLES

11819 None.

11820 APPLICATION USAGE

11821 If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared
11822 against the integer constant EOF, the comparison may never succeed, because sign-extension of
11823 a variable of type **char** on widening to integer is implementation-defined.

11824 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
11825 end-of-file condition.

11826 RATIONALE

11827 None.

11828 FUTURE DIRECTIONS

11829 None.

11830 SEE ALSO

11831 *feof()*, *ferror()*, *fopen()*, *getchar()*, *getc()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11832 <stdio.h>

11833 CHANGE HISTORY

11834 First released in Issue 1. Derived from Issue 1 of the SVID.

11835 Issue 5

11836 Large File Summit extensions are added.

11837 Issue 6

11838 Extensions beyond the ISO C standard are marked.

11839 The following new requirements on POSIX implementations derive from alignment with the
11840 Single UNIX Specification:

- 11841 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 11842 • The [ENOMEM] and [ENXIO] optional error conditions are added.

11843 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 11844 • The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the
11845 input stream is not set.
- 11846 • The RETURN VALUE section is updated to note that the error indicator shall be set for the
11847 stream.

11848 **NAME**

11849 fgetpos — get current file position information

11850 **SYNOPSIS**

11851 #include <stdio.h>

11852 int fgetpos(FILE *restrict *stream*, fpos_t *restrict *pos*);11853 **DESCRIPTION**

11854 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11855 conflict between the requirements described here and the ISO C standard is unintentional. This
 11856 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11857 The *fgetpos()* function shall store the current values of the parse state (if any) and file position
 11858 indicator for the stream pointed to by *stream* in the object pointed to by *pos*. The value stored
 11859 contains unspecified information usable by *fsetpos()* for repositioning the stream to its position
 11860 at the time of the call to *fgetpos()*.

11861 **RETURN VALUE**

11862 Upon successful completion, *fgetpos()* shall return 0; otherwise, it shall return a non-zero value
 11863 and set *errno* to indicate the error.

11864 **ERRORS**11865 The *fgetpos()* function shall fail if:

11866 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an
 11867 object of type **fpos_t**.

11868 The *fgetpos()* function may fail if:

11869 CX [EBADF] The file descriptor underlying *stream* is not valid.

11870 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.
 11871

11872 **EXAMPLES**

11873 None.

11874 **APPLICATION USAGE**

11875 None.

11876 **RATIONALE**

11877 None.

11878 **FUTURE DIRECTIONS**

11879 None.

11880 **SEE ALSO**11881 *fopen()*, *ftell()*, *rewind()*, *ungetc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>11882 **CHANGE HISTORY**

11883 First released in Issue 4. Derived from the ISO C standard.

11884 **Issue 5**

11885 Large File Summit extensions are added.

11886 **Issue 6**

11887 Extensions beyond the ISO C standard are marked.

11888 The following new requirements on POSIX implementations derive from alignment with the
 11889 Single UNIX Specification:

- 11890 • The [EBADF] and [ESPIPE] optional error conditions are added.
- 11891 An additional [ESPIPE] error condition is added for sockets.
- 11892 The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.

11893 **NAME**

11894 fgets — get a string from a stream

11895 **SYNOPSIS**

11896 #include <stdio.h>

11897 char *fgets(char *restrict *s*, int *n*, FILE *restrict *stream*);11898 **DESCRIPTION**

11899 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11900 conflict between the requirements described here and the ISO C standard is unintentional. This
 11901 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11902 The *fgets()* function shall read bytes from *stream* into the array pointed to by *s*, until *n*−1 bytes
 11903 are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered.
 11904 The string is then terminated with a null byte.

11905 CX The *fgets()* function may mark the *st_atime* field of the file associated with *stream* for update. The
 11906 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 11907 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 11908 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11909 **RETURN VALUE**

11910 Upon successful completion, *fgets()* shall return *s*. If the stream is at end-of-file, the end-of-file
 11911 indicator for the stream shall be set and *fgets()* shall return a null pointer. If a read error occurs,
 11912 CX the error indicator for the stream shall be set, *fgets()* shall return a null pointer, and shall set
 11913 *errno* to indicate the error.

11914 **ERRORS**11915 Refer to *fgetc()*.11916 **EXAMPLES**11917 **Reading Input**

11918 The following example uses *fgets()* to read each line of input. {LINE_MAX}, which defines the
 11919 maximum size of the input line, is defined in the <limits.h> header.

```
11920        #include <stdio.h>
11921        ...
11922        char line[LINE_MAX];
11923        ...
11924        while (fgets(line, LINE_MAX, fp) != NULL) {
11925        ...
11926        }
11927        ...
```

11928 **APPLICATION USAGE**

11929 None.

11930 **RATIONALE**

11931 None.

11932 **FUTURE DIRECTIONS**

11933 None.

11934 **SEE ALSO**

11935 *fopen()*, *fread()*, *gets()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

11936 **CHANGE HISTORY**

11937 First released in Issue 1. Derived from Issue 1 of the SVID.

11938 **Issue 6**

11939 Extensions beyond the ISO C standard are marked.

11940 The prototype for *fgets()* is changed for alignment with the ISO / IEC 9899: 1999 standard.

11941 **NAME**

11942 fgetwc — get a wide-character code from a stream

11943 **SYNOPSIS**

11944 #include <stdio.h>

11945 #include <wchar.h>

11946 wint_t fgetwc(FILE *stream);

11947 **DESCRIPTION**

11948 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11949 conflict between the requirements described here and the ISO C standard is unintentional. This
 11950 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11951 The *fgetwc()* function shall obtain the next character (if present) from the input stream pointed to
 11952 by *stream*, convert that to the corresponding wide-character code, and advance the associated
 11953 file position indicator for the stream (if defined).

11954 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

11955 CX The *fgetwc()* function may mark the *st_atime* field of the file associated with *stream* for update.
 11956 The *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 11957 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 11958 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11959 **RETURN VALUE**

11960 Upon successful completion, the *fgetwc()* function shall return the wide-character code of the
 11961 character read from the input stream pointed to by *stream* converted to a type **wint_t**. If the
 11962 stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetwc()* shall
 11963 return WEOF. If a read error occurs, the error indicator for the stream shall be set, *fgetwc()* shall
 11964 CX return WEOF, and shall set *errno* to indicate the error. If an encoding error occurs, the error
 11965 indicator for the stream shall be set, *fgetwc()* shall return WEOF, and shall set *errno* to indicate
 11966 the error.

11967 **ERRORS**

11968 The *fgetwc()* function shall fail if data needs to be read and:

11969 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 11970 process would be delayed in the *fgetwc()* operation.

11971 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 11972 reading.

11973 [EILSEQ] The data obtained from the input stream does not form a valid character.

11974 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 11975 was transferred.

11976 CX [EIO] A physical I/O error has occurred, or the process is in a background process
 11977 group attempting to read from its controlling terminal, and either the process
 11978 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
 11979 This error may also be generated for implementation-defined reasons.

11980 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
 11981 offset maximum associated with the corresponding stream.

11982 The *fgetwc()* function may fail if:

11983 CX [ENOMEM] Insufficient storage space is available.

11984 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 11985 capabilities of the device.

11986 **EXAMPLES**

11987 None.

11988 **APPLICATION USAGE**

11989 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
 11990 end-of-file condition.

11991 **RATIONALE**

11992 None.

11993 **FUTURE DIRECTIONS**

11994 None.

11995 **SEE ALSO**

11996 *feof()*, *ferror()*, *fopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**,
 11997 **<wchar.h>**

11998 **CHANGE HISTORY**

11999 First released in Issue 4. Derived from the MSE working draft.

12000 **Issue 5**

12001 The Optional Header (OH) marking is removed from **<stdio.h>**.

12002 Large File Summit extensions are added.

12003 **Issue 6**

12004 Extensions beyond the ISO C standard are marked.

12005 The following new requirements on POSIX implementations derive from alignment with the
 12006 Single UNIX Specification:

- 12007 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 12008 • The [ENOMEM] and [ENXIO] optional error conditions are added.

12009 **NAME**

12010 fgetws — get a wide-character string from a stream

12011 **SYNOPSIS**

12012 #include <stdio.h>

12013 #include <wchar.h>

12014 wchar_t *fgetws(wchar_t *restrict ws, int n,

12015 FILE *restrict stream);

12016 **DESCRIPTION**

12017 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 12018 conflict between the requirements described here and the ISO C standard is unintentional. This
 12019 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12020 The *fgetws()* function shall read characters from the *stream*, convert these to the corresponding
 12021 wide-character codes, place them in the **wchar_t** array pointed to by *ws*, until *n*−1 characters are
 12022 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is
 12023 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character
 12024 code.

12025 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

12026 CX The *fgetws()* function may mark the *st_atime* field of the file associated with *stream* for update.
 12027 The *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 12028 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 12029 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

12030 **RETURN VALUE**

12031 Upon successful completion, *fgetws()* shall return *ws*. If the stream is at end-of-file, the end-of-
 12032 file indicator for the stream shall be set and *fgetws()* shall return a null pointer. If a read error
 12033 CX occurs, the error indicator for the stream shall be set, *fgetws()* shall return a null pointer, and
 12034 shall set *errno* to indicate the error.

12035 **ERRORS**12036 Refer to *fgetwc()*.12037 **EXAMPLES**

12038 None.

12039 **APPLICATION USAGE**

12040 None.

12041 **RATIONALE**

12042 None.

12043 **FUTURE DIRECTIONS**

12044 None.

12045 **SEE ALSO**12046 *fopen()*, *fread()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>, <wchar.h>12047 **CHANGE HISTORY**

12048 First released in Issue 4. Derived from the MSE working draft.

12049 **Issue 5**

12050 The Optional Header (OH) marking is removed from <stdio.h>.

12051 **Issue 6**

12052 Extensions beyond the ISO C standard are marked.

12053 The prototype for *fgetws()* is changed for alignment with the ISO / IEC 9899: 1999 standard.

12054 NAME

12055 `fileno` — map a stream pointer to a file descriptor

12056 SYNOPSIS

12057 `CX` `#include <stdio.h>`

12058 `int fileno(FILE *stream);`

12059

12060 DESCRIPTION

12061 The `fileno()` function shall return the integer file descriptor associated with the stream pointed to
12062 by `stream`.

12063 RETURN VALUE

12064 Upon successful completion, `fileno()` shall return the integer value of the file descriptor
12065 associated with `stream`. Otherwise, the value `-1` shall be returned and `errno` set to indicate the
12066 error.

12067 ERRORS

12068 The `fileno()` function may fail if:

12069 [EBADF] The `stream` argument is not a valid stream.

12070 EXAMPLES

12071 None.

12072 APPLICATION USAGE

12073 None.

12074 RATIONALE

12075 Without some specification of which file descriptors are associated with these streams, it is
12076 impossible for an application to set up the streams for another application it starts with `fork()`
12077 and `exec`. In particular, it would not be possible to write a portable version of the `sh` command
12078 interpreter (although there may be other constraints that would prevent that portability).

12079 FUTURE DIRECTIONS

12080 None.

12081 SEE ALSO

12082 Section 2.5.1 (on page 35), `fdopen()`, `fopen()`, `stdin`, the Base Definitions volume of
12083 IEEE Std 1003.1-2001, `<stdio.h>`

12084 CHANGE HISTORY

12085 First released in Issue 1. Derived from Issue 1 of the SVID.

12086 Issue 6

12087 The following new requirements on POSIX implementations derive from alignment with the
12088 Single UNIX Specification:

- 12089 • The [EBADF] optional error condition is added.

12090 **NAME**

12091 flockfile, ftrylockfile, funlockfile — stdio locking functions

12092 **SYNOPSIS**12093 TSF `#include <stdio.h>`

```
12094 void flockfile(FILE *file);
12095 int ftrylockfile(FILE *file);
12096 void funlockfile(FILE *file);
12097
```

12098 **DESCRIPTION**

12099 These functions shall provide for explicit application-level locking of stdio (**FILE ***) objects.
 12100 These functions can be used by a thread to delineate a sequence of I/O statements that are
 12101 executed as a unit.

12102 The *flockfile()* function shall acquire for a thread ownership of a (**FILE ***) object.

12103 The *ftrylockfile()* function shall acquire for a thread ownership of a (**FILE ***) object if the object is
 12104 available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

12105 The *funlockfile()* function shall relinquish the ownership granted to the thread. The behavior is
 12106 undefined if a thread other than the current owner calls the *funlockfile()* function.

12107 The functions shall behave as if there is a lock count associated with each (**FILE ***) object. This
 12108 count is implicitly initialized to zero when the (**FILE ***) object is created. The (**FILE ***) object is
 12109 unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE ***)
 12110 object. When the *flockfile()* function is called, if the count is zero or if the count is positive and
 12111 the caller owns the (**FILE ***) object, the count shall be incremented. Otherwise, the calling thread
 12112 shall be suspended, waiting for the count to return to zero. Each call to *funlockfile()* shall
 12113 decrement the count. This allows matching calls to *flockfile()* (or successful calls to *ftrylockfile()*)
 12114 and *funlockfile()* to be nested.

12115 All functions that reference (**FILE ***) objects shall behave as if they use *flockfile()* and *funlockfile()*
 12116 internally to obtain ownership of these (**FILE ***) objects.

12117 **RETURN VALUE**

12118 None for *flockfile()* and *funlockfile()*.

12119 The *ftrylockfile()* function shall return zero for success and non-zero to indicate that the lock
 12120 cannot be acquired.

12121 **ERRORS**

12122 No errors are defined.

12123 **EXAMPLES**

12124 None.

12125 **APPLICATION USAGE**

12126 Applications using these functions may be subject to priority inversion, as discussed in the Base
 12127 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

12128 **RATIONALE**

12129 The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual-exclusion lock for each
 12130 **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a file lock,
 12131 analogous to *pthread_mutex_trylock()*.

12132 These locks behave as if they are the same as those used internally by *stdio* for thread-safety.
 12133 This both provides thread-safety of these functions without requiring a second level of internal
 12134 locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

12135 Application writers and implementors should be aware that there are potential deadlock
12136 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested
12137 via `{_IOLBF}`) require that certain input operations sometimes cause the buffered contents of
12138 implementation-defined line-buffered output streams to be flushed. If two threads each hold the
12139 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring
12140 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can
12141 typically be avoided by acquiring locks on input streams before locks on output streams if a
12142 thread would be acquiring both.

12143 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*
12144 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where
12145 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the
12146 `*_unlocked()` functions/macros. This moves the cost/performance tradeoff to the optimal point.

12147 **FUTURE DIRECTIONS**

12148 None.

12149 **SEE ALSO**

12150 *getc_unlocked()*, *putc_unlocked()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdio.h>`

12151 **CHANGE HISTORY**

12152 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

12153 **Issue 6**

12154 These functions are marked as part of the Thread-Safe Functions option.

12155 **NAME**

12156 floor, floorf, floorl — floor function

12157 **SYNOPSIS**

12158 #include <math.h>

12159 double floor(double x);

12160 float floorf(float x);

12161 long double floorl(long double x);

12162 **DESCRIPTION**

12163 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 12164 conflict between the requirements described here and the ISO C standard is unintentional. This
 12165 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12166 These functions shall compute the largest integral value not greater than *x*.

12167 An application wishing to check for error situations should set *errno* to zero and call
 12168 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 12169 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 12170 zero, an error has occurred.

12171 **RETURN VALUE**

12172 Upon successful completion, these functions shall return the largest integral value not greater
 12173 than *x*, expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the
 12174 function.

12175 MX If *x* is NaN, a NaN shall be returned.12176 If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.

12177 XSI If the correct value would cause overflow, a range error shall occur and *floor*(), *floorf*(), and
 12178 *floorl*() shall return the value of the macro `-HUGE_VAL`, `-HUGE_VALF`, and `-HUGE_VALL`,
 12179 respectively.

12180 **ERRORS**

12181 These functions shall fail if:

12182 XSI **Range Error** The result would cause an overflow.

12183 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12184 then *errno* shall be set to [ERANGE]. If the integer expression
 12185 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 12186 floating-point exception shall be raised.

12187 **EXAMPLES**

12188 None.

12189 **APPLICATION USAGE**

12190 The integral value returned by these functions might not be expressible as an **int** or **long**. The
 12191 return value should be tested before assigning it to an integer type to avoid the undefined results
 12192 of an integer overflow.

12193 The *floor*() function can only overflow when the floating-point representation has
 12194 DBL_MANT_DIG > DBL_MAX_EXP.

12195 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 12196 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

12197 **RATIONALE**

12198 None.

12199 **FUTURE DIRECTIONS**

12200 None.

12201 **SEE ALSO**

12202 *ceil()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
12203 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

12204 **CHANGE HISTORY**

12205 First released in Issue 1. Derived from Issue 1 of the SVID.

12206 **Issue 5**

12207 The DESCRIPTION is updated to indicate how an application should check for an error. This
12208 text was previously published in the APPLICATION USAGE section.

12209 **Issue 6**

12210 The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

12211 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
12212 revised to align with the ISO/IEC 9899:1999 standard.

12213 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
12214 marked.

12215 **NAME**

12216 fma, fmaf, fmal — floating-point multiply-add

12217 **SYNOPSIS**

12218 #include <math.h>

12219 double fma(double x, double y, double z);

12220 float fmaf(float x, float y, float z);

12221 long double fmal(long double x, long double y, long double z);

12222 **DESCRIPTION**

12223 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 12224 conflict between the requirements described here and the ISO C standard is unintentional. This
 12225 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12226 These functions shall compute $(x * y) + z$, rounded as one ternary operation: they shall compute
 12227 the value (as if) to infinite precision and round once to the result format, according to the
 12228 rounding mode characterized by the value of FLT_ROUNDS.

12229 An application wishing to check for error situations should set *errno* to zero and call
 12230 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 12231 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 12232 zero, an error has occurred.

12233 **RETURN VALUE**

12234 Upon successful completion, these functions shall return $(x * y) + z$, rounded as one ternary
 12235 operation.

12236 MX If *x* or *y* are NaN, a NaN shall be returned.

12237 If *x* multiplied by *y* is an exact infinity and *z* is also an infinity but with the opposite sign, a
 12238 domain error shall occur, and either a NaN (if supported), or an implementation-defined value
 12239 shall be returned.

12240 If one of *x* and *y* is infinite, the other is zero, and *z* is not a NaN, a domain error shall occur, and
 12241 either a NaN (if supported), or an implementation-defined value shall be returned.

12242 If one of *x* and *y* is infinite, the other is zero, and *z* is a NaN, a NaN shall be returned and a
 12243 domain error may occur.

12244 If $x * y$ is not $0 * \text{Inf}$ nor $\text{Inf} * 0$ and *z* is a NaN, a NaN shall be returned.

12245 **ERRORS**

12246 These functions shall fail if:

12247 MX **Domain Error** The value of $x * y + z$ is invalid, or the value $x * y$ is invalid and *z* is not a NaN.

12248 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12249 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 12250 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 12251 shall be raised.

12252 MX **Range Error** The result overflows.

12253 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12254 then *errno* shall be set to [ERANGE]. If the integer expression
 12255 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 12256 floating-point exception shall be raised.

12257 These functions may fail if:

12258 MX **Domain Error** The value $x*y$ is invalid and z is a NaN.

12259 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,
 12260 then *errno* shall be set to [EDOM]. If the integer expression (`math_errhandling`
 12261 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception
 12262 shall be raised.

12263 MX **Range Error** The result underflows.

12264 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,
 12265 then *errno* shall be set to [ERANGE]. If the integer expression
 12266 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the underflow
 12267 floating-point exception shall be raised.

12268 EXAMPLES

12269 None.

12270 APPLICATION USAGE

12271 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
 12272 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

12273 RATIONALE

12274 In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its
 12275 unexpected use by the compiler can undermine carefully written code. The `FP_CONTRACT`
 12276 macro can be used to disallow use of floating multiply-add; and the *fma()* function guarantees
 12277 its use where desired. Many current machines provide hardware floating multiply-add
 12278 instructions; software implementation can be used for others.

12279 FUTURE DIRECTIONS

12280 None.

12281 SEE ALSO

12282 *feclearexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,
 12283 Treatment of Error Conditions for Mathematical Functions, <math.h>

12284 CHANGE HISTORY

12285 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12286 **NAME**

12287 fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

12288 **SYNOPSIS**

12289 #include <math.h>

12290 double fmax(double x, double y);

12291 float fmaxf(float x, float y);

12292 long double fmaxl(long double x, long double y);

12293 **DESCRIPTION**

12294 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12295 conflict between the requirements described here and the ISO C standard is unintentional. This
12296 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12297 These functions shall determine the maximum numeric value of their arguments. NaN
12298 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
12299 then these functions shall choose the numeric value.

12300 **RETURN VALUE**

12301 Upon successful completion, these functions shall return the maximum numeric value of their
12302 arguments.

12303 If just one argument is a NaN, the other argument shall be returned.

12304 MX If x and y are NaN, a NaN shall be returned.

12305 **ERRORS**

12306 No errors are defined.

12307 **EXAMPLES**

12308 None.

12309 **APPLICATION USAGE**

12310 None.

12311 **RATIONALE**

12312 None.

12313 **FUTURE DIRECTIONS**

12314 None.

12315 **SEE ALSO**

12316 *fdim()*, *fmin()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

12317 **CHANGE HISTORY**

12318 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12319 **NAME**

12320 fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

12321 **SYNOPSIS**

12322 #include <math.h>

12323 double fmin(double x, double y);

12324 float fminf(float x, float y);

12325 long double fminl(long double x, long double y);

12326 **DESCRIPTION**

12327 CX The functionality described on this reference page is aligned with the ISO C standard. Any
12328 conflict between the requirements described here and the ISO C standard is unintentional. This
12329 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12330 These functions shall determine the minimum numeric value of their arguments. NaN
12331 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
12332 then these functions shall choose the numeric value.

12333 **RETURN VALUE**

12334 Upon successful completion, these functions shall return the minimum numeric value of their
12335 arguments.

12336 If just one argument is a NaN, the other argument shall be returned.

12337 MX If x and y are NaN, a NaN shall be returned.

12338 **ERRORS**

12339 No errors are defined.

12340 **EXAMPLES**

12341 None.

12342 **APPLICATION USAGE**

12343 None.

12344 **RATIONALE**

12345 None.

12346 **FUTURE DIRECTIONS**

12347 None.

12348 **SEE ALSO**

12349 *fdim()*, *fmax()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

12350 **CHANGE HISTORY**

12351 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12352 **NAME**

12353 fmod, fmodf, fmodl — floating-point remainder value function

12354 **SYNOPSIS**

12355 #include <math.h>

12356 double fmod(double x, double y);

12357 float fmodf(float x, float y);

12358 long double fmodl(long double x, long double y);

12359 **DESCRIPTION**

12360 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 12361 conflict between the requirements described here and the ISO C standard is unintentional. This
 12362 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12363 These functions shall return the floating-point remainder of the division of x by y .

12364 An application wishing to check for error situations should set *errno* to zero and call
 12365 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 12366 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 12367 zero, an error has occurred.

12368 **RETURN VALUE**

12369 These functions shall return the value $x - i*y$, for some integer i such that, if y is non-zero, the
 12370 result has the same sign as x and magnitude less than the magnitude of y .

12371 If the correct value would cause underflow, and is not representable, a range error may occur,
 12372 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

12373 MX If x or y is NaN, a NaN shall be returned.

12374 If y is zero, a domain error shall occur, and either a NaN (if supported), or an implementation-
 12375 defined value shall be returned.

12376 If x is infinite, a domain error shall occur, and either a NaN (if supported), or an
 12377 implementation-defined value shall be returned.

12378 If x is ± 0 and y is not zero, ± 0 shall be returned.12379 If x is not infinite and y is $\pm \text{Inf}$, x shall be returned.

12380 If the correct value would cause underflow, and is representable, a range error may occur and
 12381 the correct value shall be returned.

12382 **ERRORS**

12383 These functions shall fail if:

12384 MX **Domain Error** The x argument is infinite or y is zero.

12385 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12386 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 12387 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 12388 shall be raised.

12389 These functions may fail if:

12390 **Range Error** The result underflows.

12391 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12392 then *errno* shall be set to [ERANGE]. If the integer expression
 12393 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 12394 floating-point exception shall be raised.

12395 EXAMPLES

12396 None.

12397 APPLICATION USAGE

12398 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
12399 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

12400 RATIONALE

12401 None.

12402 FUTURE DIRECTIONS

12403 None.

12404 SEE ALSO

12405 *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
12406 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**>

12407 CHANGE HISTORY

12408 First released in Issue 1. Derived from Issue 1 of the SVID.

12409 Issue 5

12410 The DESCRIPTION is updated to indicate how an application should check for an error. This
12411 text was previously published in the APPLICATION USAGE section.

12412 Issue 6

12413 The behavior for when the *y* argument is zero is now defined.

12414 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999
12415 standard.

12416 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
12417 revised to align with the ISO/IEC 9899:1999 standard.

12418 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
12419 marked.

12420 **NAME**

12421 fmtmsg — display a message in the specified format on standard error and/or a system console

12422 **SYNOPSIS**

12423 xSI #include <fmtmsg.h>

```
12424       int fmtmsg(long classification, const char *label, int severity,
12425               const char *text, const char *action, const char *tag);
```

12426

12427 **DESCRIPTION**12428 The *fmtmsg()* function shall display messages in a specified format instead of the traditional
12429 *printf()* function.12430 Based on a message's classification component, *fmtmsg()* shall write a formatted message either
12431 to standard error, to the console, or to both.12432 A formatted message consists of up to five components as defined below. The component
12433 *classification* is not part of a message displayed to the user, but defines the source of the message
12434 and directs the display of the formatted message.

12435 *classification* Contains the sum of identifying values constructed from the constants defined
12436 below. Any one identifier from a subclass may be used in combination with a
12437 single identifier from a different subclass. Two or more identifiers from the
12438 same subclass should not be used together, with the exception of identifiers
12439 from the display subclass. (Both display subclass identifiers may be used so
12440 that messages can be displayed to both standard error and the system
12441 console.)

12442 **Major Classifications**12443 Identifies the source of the condition. Identifiers are: MM_HARD
12444 (hardware), MM_SOFT (software), and MM_FIRM (firmware).12445 **Message Source Subclassifications**12446 Identifies the type of software in which the problem is detected.
12447 Identifiers are: MM_APPL (application), MM_UTIL (utility), and
12448 MM_OPSYS (operating system).12449 **Display Subclassifications**12450 Indicates where the message is to be displayed. Identifiers are:
12451 MM_PRINT to display the message on the standard error stream,
12452 MM_CONSOLE to display the message on the system console. One or
12453 both identifiers may be used.12454 **Status Subclassifications**12455 Indicates whether the application can recover from the condition.
12456 Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-
12457 recoverable).12458 An additional identifier, MM_NULLMC, indicates that no classification
12459 component is supplied for the message.12460 *label* Identifies the source of the message. The format is two fields separated by a
12461 colon. The first field is up to 10 bytes, the second is up to 14 bytes.12462 *severity* Indicates the seriousness of the condition. Identifiers for the levels of *severity*
12463 are:

12464		MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
12465			
12466		MM_ERROR	Indicates that the application has detected a fault. Produces the string "ERROR".
12467			
12468		MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".
12469			
12470			
12471		MM_INFO	Provides information about a condition that is not in error. Produces the string "INFO".
12472			
12473		MM_NOSEV	Indicates that no severity level is supplied for the message.
12474	<i>text</i>		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
12475			
12476			
12477	<i>action</i>		Describes the first step to be taken in the error-recovery process. The <i>fmtmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size.
12478			
12479			
12480	<i>tag</i>		An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".
12481			
12482			

12483 The *MSGVERB* environment variable (for message verbosity) shall determine for *fmtmsg()*
 12484 which message components it is to select when writing messages to standard error. The value of
 12485 *MSGVERB* shall be a colon-separated list of optional keywords. Valid keywords are: *label*,
 12486 *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the
 12487 component's value is not the component's null value, *fmtmsg()* shall include that component in
 12488 the message when writing the message to standard error. If *MSGVERB* does not include a
 12489 keyword for a message component, that component shall not be included in the display of the
 12490 message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the
 12491 null string, if its value is not of the correct format, or if it contains keywords other than the valid
 12492 ones listed above, *fmtmsg()* shall select all components.

12493 *MSGVERB* shall determine which components are selected for display to standard error. All
 12494 message components shall be included in console messages.

12495 RETURN VALUE

12496 The *fmtmsg()* function shall return one of the following values:

12497	MM_OK	The function succeeded.
12498	MM_NOTOK	The function failed completely.
12499	MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.
12500		
12501	MM_NOCON	The function was unable to generate a console message, but otherwise succeeded.
12502		

12503 ERRORS

12504 None.

12505 **EXAMPLES**

12506 1. The following example of *fmtmsg()*:

12507 `fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",`
12508 `"refer to cat in user's reference manual", "XSI:cat:001")`

12509 produces a complete message in the specified message format:

12510 `XSI:cat: ERROR: illegal option`
12511 `TO FIX: refer to cat in user's reference manual XSI:cat:001`

12512 2. When the environment variable *MSGVERB* is set as follows:

12513 `MSGVERB=severity:text:action`

12514 and Example 1 is used, *fmtmsg()* produces:

12515 `ERROR: illegal option`
12516 `TO FIX: refer to cat in user's reference manual`

12517 **APPLICATION USAGE**

12518 One or more message components may be systematically omitted from messages generated by
12519 an application by using the null value of the argument for that component.

12520 **RATIONALE**

12521 None.

12522 **FUTURE DIRECTIONS**

12523 None.

12524 **SEE ALSO**

12525 *printf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**fmtmsg.h**>

12526 **CHANGE HISTORY**

12527 First released in Issue 4, Version 2.

12528 **Issue 5**

12529 Moved from X/OPEN UNIX extension to BASE.

12530 **NAME**

12531 fnmatch — match a filename or a pathname

12532 **SYNOPSIS**

12533 #include <fnmatch.h>

12534 int fnmatch(const char **pattern*, const char **string*, int *flags*);12535 **DESCRIPTION**

12536 The *fnmatch()* function shall match patterns as described in the Shell and Utilities volume of
 12537 IEEE Std 1003.1-2001, Section 2.13.1, Patterns Matching a Single Character, and Section 2.13.2,
 12538 Patterns Matching Multiple Characters. It checks the string specified by the *string* argument to
 12539 see if it matches the pattern specified by the *pattern* argument.

12540 The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-inclusive
 12541 OR of zero or more of the flags defined in <fnmatch.h>. If the FNM_PATHNAME flag is set in
 12542 *flags*, then a slash character ('/') in *string* shall be explicitly matched by a slash in *pattern*; it shall
 12543 not be matched by either the asterisk or question-mark special characters, nor by a bracket
 12544 expression. If the FNM_PATHNAME flag is not set, the slash character shall be treated as an
 12545 ordinary character.

12546 If FNM_NOESCAPE is not set in *flags*, a backslash character ('\ ') in *pattern* followed by any
 12547 other character shall match that second character in *string*. In particular, "\\\" shall match a
 12548 backslash in *string*. If FNM_NOESCAPE is set, a backslash character shall be treated as an
 12549 ordinary character.

12550 If FNM_PERIOD is set in *flags*, then a leading period (' . ') in *string* shall match a period in
 12551 *pattern*; as described by rule 2 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
 12552 2.13.3, Patterns Used for Filename Expansion where the location of “leading” is indicated by the
 12553 value of FNM_PATHNAME:

- 12554 • If FNM_PATHNAME is set, a period is “leading” if it is the first character in *string* or if it
 12555 immediately follows a slash.
- 12556 • If FNM_PATHNAME is not set, a period is “leading” only if it is the first character of *string*.

12557 If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

12558 **RETURN VALUE**

12559 If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return 0. If there is no
 12560 match, *fnmatch()* shall return FNM_NOMATCH, which is defined in <fnmatch.h>. If an error
 12561 occurs, *fnmatch()* shall return another non-zero value.

12562 **ERRORS**

12563 No errors are defined.

12564 **EXAMPLES**

12565 None.

12566 **APPLICATION USAGE**

12567 The *fnmatch()* function has two major uses. It could be used by an application or utility that
 12568 needs to read a directory and apply a pattern against each entry. The *find* utility is an example of
 12569 this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that
 12570 need to match strings in a similar manner.

12571 The name *fnmatch()* is intended to imply *filename* match, rather than *pathname* match. The default
 12572 action of this function is to match filenames, rather than pathnames, since it gives no special
 12573 significance to the slash character. With the FNM_PATHNAME flag, *fnmatch()* does match
 12574 pathnames, but without tilde expansion, parameter expansion, or special treatment for a period

12575 at the beginning of a filename.

12576 **RATIONALE**

12577 This function replaced the REG_FILENAME flag of *regcomp()* in early proposals of this volume
12578 of IEEE Std 1003.1-2001. It provides virtually the same functionality as the *regcomp()* and
12579 *regexexec()* functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag
12580 was proposed for *regcomp()*, and would have had the opposite effect from FNM_PATHNAME),
12581 but with a simpler function and less system overhead.

12582 **FUTURE DIRECTIONS**

12583 None.

12584 **SEE ALSO**

12585 *glob()*, *wordexp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**fnmatch.h**>, the Shell
12586 and Utilities volume of IEEE Std 1003.1-2001

12587 **CHANGE HISTORY**

12588 First released in Issue 4. Derived from the ISO POSIX-2 standard.

12589 **Issue 5**

12590 Moved from POSIX2 C-language Binding to BASE.

12591 **NAME**

12592 fopen — open a stream

12593 **SYNOPSIS**

12594 #include <stdio.h>

12595 FILE *fopen(const char *restrict *filename*, const char *restrict *mode*);12596 **DESCRIPTION**

12597 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 12598 conflict between the requirements described here and the ISO C standard is unintentional. This
 12599 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12600 The *fopen()* function shall open the file whose pathname is the string pointed to by *filename*, and
 12601 associates a stream with it.

12602 The *mode* argument points to a string. If the string is one of the following, the file shall be opened
 12603 in the indicated mode. Otherwise, the behavior is undefined.

12604 *r* or *rb* Open file for reading.

12605 *w* or *wb* Truncate to zero length or create file for writing.

12606 *a* or *ab* Append; open or create file for writing at end-of-file.

12607 *r+* or *rb+* or *r+b* Open file for update (reading and writing).

12608 *w+* or *wb+* or *w+b* Truncate to zero length or create file for update.

12609 *a+* or *ab+* or *a+b* Append; open or create file for update, writing at end-of-file.

12610 CX The character 'b' shall have no effect, but is allowed for ISO C standard conformance. Opening
 12611 a file with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not
 12612 exist or cannot be read.

12613 Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all
 12614 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening
 12615 calls to *fseek()*.

12616 When a file is opened with update mode ('+' as the second or third character in the *mode*
 12617 argument), both input and output may be performed on the associated stream. However, the
 12618 application shall ensure that output is not directly followed by input without an intervening call
 12619 to *fflush()* or to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*), and input is not directly
 12620 followed by output without an intervening call to a file positioning function, unless the input
 12621 operation encounters end-of-file.

12622 When opened, a stream is fully buffered if and only if it can be determined not to refer to an
 12623 interactive device. The error and end-of-file indicators for the stream shall be cleared.

12624 CX If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, upon
 12625 successful completion, the *fopen()* function shall mark for update the *st_atime*, *st_ctime*, and
 12626 *st_mtime* fields of the file and the *st_ctime* and *st_mtime* fields of the parent directory.

12627 If *mode* is *w*, *wb*, *w+*, *wb+*, or *w+b*, and the file did previously exist, upon successful completion,
 12628 *fopen()* shall mark for update the *st_ctime* and *st_mtime* fields of the file. The *fopen()* function
 12629 shall allocate a file descriptor as *open()* does.

12630 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the
 12631 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an
 12632 initial conversion state.

12633 CX The largest value that can be represented correctly in an object of type **off_t** shall be established
 12634 as the offset maximum in the open file description.

12635 RETURN VALUE

12636 Upon successful completion, *fopen()* shall return a pointer to the object controlling the stream.
 12637 CX Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

12638 ERRORS

12639 The *fopen()* function shall fail if:

12640 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
 12641 exists and the permissions specified by *mode* are denied, or the file does not
 12642 exist and write permission is denied for the parent directory of the file to be
 12643 created.

12644 CX [EINTR] A signal was caught during *fopen()*.

12645 CX [EISDIR] The named file is a directory and *mode* requires write access.

12646 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 12647 argument.

12648 CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

12649 CX [ENAMETOOLONG] The length of the *filename* argument exceeds {PATH_MAX} or a pathname
 12650 component is longer than {NAME_MAX}.
 12651

12652 CX [ENFILE] The maximum allowable number of files is currently open in the system.

12653 CX [ENOENT] A component of *filename* does not name an existing file or *filename* is an empty
 12654 string.

12655 CX [ENOSPC] The directory or file system that would contain the new file cannot be
 12656 expanded, the file does not exist, and the file was to be created.

12657 CX [ENOTDIR] A component of the path prefix is not a directory.

12658 CX [ENXIO] The named file is a character special or block special file, and the device
 12659 associated with this special file does not exist.

12660 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented
 12661 correctly in an object of type **off_t**.

12662 CX [EROFS] The named file resides on a read-only file system and *mode* requires write
 12663 access.

12664 The *fopen()* function may fail if:

12665 CX [EINVAL] The value of the *mode* argument is not valid.

12666 CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 12667 resolution of the *path* argument.

12668 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

12669 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

12670 CX [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result
 12671 whose length exceeds {PATH_MAX}.
 12672

12673	CX	[ENOMEM]	Insufficient storage space is available.
12674	CX	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i>
12675			requires write access.

12676 **EXAMPLES**12677 **Opening a File**

12678 The following example tries to open the file named **file** for reading. The *fopen()* function returns
 12679 a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the
 12680 file, it just ignores it.

```

12681 #include <stdio.h>
12682 ...
12683 FILE *fp;
12684 ...
12685 void rgrep(const char *file)
12686 {
12687     ...
12688     if ((fp = fopen(file, "r")) == NULL)
12689         return;
12690     ...
12691 }
```

12692 **APPLICATION USAGE**

12693 None.

12694 **RATIONALE**

12695 None.

12696 **FUTURE DIRECTIONS**

12697 None.

12698 **SEE ALSO**

12699 *fclose()*, *fdopen()*, *freopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

12700 **CHANGE HISTORY**

12701 First released in Issue 1. Derived from Issue 1 of the SVID.

12702 **Issue 5**

12703 Large File Summit extensions are added.

12704 **Issue 6**

12705 Extensions beyond the ISO C standard are marked.

12706 The following new requirements on POSIX implementations derive from alignment with the
 12707 Single UNIX Specification:

- 12708 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
 12709 description. This change is to support large files.
- 12710 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
 12711 large files.
- 12712 • The [ELOOP] mandatory error condition is added.
- 12713 • The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional error
 12714 conditions are added.

- 12715 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 12716 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 12717 • The prototype for *fopen()* is updated.
 - 12718 • The DESCRIPTION is updated to note that if the argument *mode* points to a string other than
 - 12719 those listed, then the behavior is undefined.
- 12720 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
- 12721 [ELOOP] error condition is added.

12722 **NAME**

12723 fork — create a new process

12724 **SYNOPSIS**

12725 #include <unistd.h>

12726 pid_t fork(void);

12727 **DESCRIPTION**

12728 The *fork()* function shall create a new process. The new process (child process) shall be an exact
 12729 copy of the calling process (parent process) except as detailed below:

- 12730 • The child process shall have a unique process ID.
- 12731 • The child process ID also shall not match any active process group ID.
- 12732 • The child process shall have a different parent process ID, which shall be the process ID of
 12733 the calling process.
- 12734 • The child process shall have its own copy of the parent's file descriptors. Each of the child's
 12735 file descriptors shall refer to the same open file description with the corresponding file
 12736 descriptor of the parent.
- 12737 • The child process shall have its own copy of the parent's open directory streams. Each open
 12738 directory stream in the child process may share directory stream positioning with the
 12739 corresponding directory stream of the parent.
- 12740 XSI • The child process shall have its own copy of the parent's message catalog descriptors.
- 12741 • The child process' values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* shall be set to 0.
- 12742 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be
 12743 canceled; see *alarm()*.
- 12744 XSI • All *semadj* values shall be cleared.
- 12745 • File locks set by the parent process shall not be inherited by the child process.
- 12746 • The set of signals pending for the child process shall be initialized to the empty set.
- 12747 XSI • Interval timers shall be reset in the child process.
- 12748 SEM • Any semaphores that are open in the parent process shall also be open in the child process.
- 12749 ML • The child process shall not inherit any address space memory locks established by the parent
 12750 process via calls to *mlockall()* or *mlock()*.
- 12751 MF|SHM • Memory mappings created in the parent shall be retained in the child process.
 12752 MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE mappings
 12753 in the child, and any modifications to the data in these mappings made by the parent prior to
 12754 calling *fork()* shall be visible to the child. Any modifications to the data in MAP_PRIVATE
 12755 mappings made by the parent after *fork()* returns shall be visible only to the parent.
 12756 Modifications to the data in MAP_PRIVATE mappings made by the child shall be visible only
 12757 to the child.
- 12758 PS • For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the
 12759 policy and priority settings of the parent process during a *fork()* function. For other
 12760 scheduling policies, the policy and priority settings on *fork()* are implementation-defined.
- 12761 TMR • Per-process timers created by the parent shall not be inherited by the child process.
- 12762 MSG • The child process shall have its own copy of the message queue descriptors of the parent.
 12763 Each of the message descriptors of the child shall refer to the same open message queue

12764		description as the corresponding message descriptor of the parent.
12765 AIO		• No asynchronous input or asynchronous output operations shall be inherited by the child
12766		process.
12767		• A process shall be created with a single thread. If a multi-threaded process calls <i>fork()</i> , the
12768		new process shall contain a replica of the calling thread and its entire address space, possibly
12769		including the states of mutexes and other resources. Consequently, to avoid errors, the child
12770		process may only execute async-signal-safe operations until such time as one of the <i>exec</i>
12771 THR		functions is called. Fork handlers may be established by means of the <i>pthread_atfork()</i>
12772		function in order to maintain application invariants across <i>fork()</i> calls.
12773		When the application calls <i>fork()</i> from a signal handler and any of the fork handlers
12774		registered by <i>pthread_atfork()</i> calls a function that is not asynch-signal-safe, the behavior is
12775		undefined.
12776 TRC TRI		• If the Trace option and the Trace Inherit option are both supported:
12777		If the calling process was being traced in a trace stream that had its inheritance policy set to
12778		POSIX_TRACE_INHERITED, the child process shall be traced into that trace stream, and the
12779		child process shall inherit the parent's mapping of trace event names to trace event type
12780		identifiers. If the trace stream in which the calling process was being traced had its
12781		inheritance policy set to POSIX_TRACE_CLOSE_FOR_CHILD, the child process shall not be
12782		traced into that trace stream. The inheritance policy is set by a call to the
12783		<i>posix_trace_attr_setinherited()</i> function.
12784 TRC		• If the Trace option is supported, but the Trace Inherit option is not supported:
12785		The child process shall not be traced into any of the trace streams of its parent process.
12786 TRC		• If the Trace option is supported, the child process of a trace controller process shall not
12787		control the trace streams controlled by its parent process.
12788 CPT		• The initial value of the CPU-time clock of the child process shall be set to zero.
12789 TCT		• The initial value of the CPU-time clock of the single thread of the child process shall be set to
12790		zero.
12791		All other process characteristics defined by IEEE Std 1003.1-2001 shall be the same in the parent
12792		and child processes. The inheritance of process characteristics not defined by
12793		IEEE Std 1003.1-2001 is unspecified by IEEE Std 1003.1-2001.
12794		After <i>fork()</i> , both the parent and the child processes shall be capable of executing independently
12795		before either one terminates.
12796	RETURN VALUE	
12797		Upon successful completion, <i>fork()</i> shall return 0 to the child process and shall return the
12798		process ID of the child process to the parent process. Both processes shall continue to execute
12799		from the <i>fork()</i> function. Otherwise, -1 shall be returned to the parent process, no child process
12800		shall be created, and <i>errno</i> shall be set to indicate the error.
12801	ERRORS	
12802		The <i>fork()</i> function shall fail if:
12803	[EAGAIN]	The system lacked the necessary resources to create another process, or the
12804		system-imposed limit on the total number of processes under execution
12805		system-wide or by a single user {CHILD_MAX} would be exceeded.

12806 The *fork()* function may fail if:

12807 [ENOMEM] Insufficient storage space is available.

12808 **EXAMPLES**

12809 None.

12810 **APPLICATION USAGE**

12811 None.

12812 **RATIONALE**

12813 Many historical implementations have timing windows where a signal sent to a process group
 12814 (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the
 12815 parent following the *fork()* but not to the child because the *fork()* code clears the child's set of
 12816 pending signals. This volume of IEEE Std 1003.1-2001 does not require, or even permit, this
 12817 behavior. However, it is pragmatic to expect that problems of this nature may continue to exist
 12818 in implementations that appear to conform to this volume of IEEE Std 1003.1-2001 and pass
 12819 available verification suites. This behavior is only a consequence of the implementation failing to
 12820 make the interval between signal generation and delivery totally invisible. From the
 12821 application's perspective, a *fork()* call should appear atomic. A signal that is generated prior to
 12822 the *fork()* should be delivered prior to the *fork()*. A signal sent to the process group after the
 12823 *fork()* should be delivered to both parent and child. The implementation may actually initialize
 12824 internal data structures corresponding to the child's set of pending signals to include signals
 12825 sent to the process group during the *fork()*. Since the *fork()* call can be considered as atomic
 12826 from the application's perspective, the set would be initialized as empty and such signals would
 12827 have arrived after the *fork()*; see also **<signal.h>**.

12828 One approach that has been suggested to address the problem of signal inheritance across *fork()*
 12829 is to add an [EINTR] error, which would be returned when a signal is detected during the call.
 12830 While this is preferable to losing signals, it was not considered an optimal solution. Although it
 12831 is not recommended for this purpose, such an error would be an allowable extension for an
 12832 implementation.

12833 The [ENOMEM] error value is reserved for those implementations that detect and distinguish
 12834 such a condition. This condition occurs when an implementation detects that there is not enough
 12835 memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate
 12836 because there can never be enough memory (either primary or secondary storage) to perform the
 12837 operation. Since *fork()* duplicates an existing process, this must be a condition where there is
 12838 sufficient memory for one such process, but not for two. Many historical implementations
 12839 actually return [ENOMEM] due to temporary lack of memory, a case that is not generally
 12840 distinct from [EAGAIN] from the perspective of a conforming application.

12841 Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it
 12842 and it should be reserved for the error condition specified there. The condition is not applicable
 12843 on many implementations.

12844 IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*.
 12845 A system that single-threads processes was clearly not intended and is considered an
 12846 unacceptable "toy implementation" of this volume of IEEE Std 1003.1-2001. The only objection
 12847 anticipated to the phrase "executing independently" is testability, but this assertion should be
 12848 testable. Such tests require that both the parent and child can block on a detectable action of the
 12849 other, such as a write to a pipe or a signal. An interactive exchange of such actions should be
 12850 possible for the system to conform to the intent of this volume of IEEE Std 1003.1-2001.

12851 The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it
 12852 occurs or not is not in any practical sense under the control of the application because the
 12853 condition is usually a consequence of the user's use of the system, not of the application's code.

Thus, no application can or should rely upon its occurrence under any circumstances, nor should the exact semantics of what concept of “user” is used be of concern to the application writer. Validation writers should be cognizant of this limitation.

There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread of control within the same program (which was originally only possible in POSIX by creating a new process); the other is to create a new process running a different program. In the latter case, the call to *fork()* is soon followed by a call to one of the *exec* functions.

The general problem with making *fork()* work in a multi-threaded world is what to do with all of the threads. There are two alternatives. One is to copy all of the threads into the new process. This causes the programmer or implementation to deal with threads that are suspended on system calls or that might be about to execute system calls that should not be executed in the new process. The other alternative is to copy only the thread that calls *fork()*. This creates the difficulty that the state of process-local resources is usually held in process memory. If a thread that is not calling *fork()* holds a resource, that resource is never released in the child process because the thread whose job it is to release the resource does not exist in the child process.

When a programmer is writing a multi-threaded program, the first described use of *fork()*, creating new threads in the same program, is provided by the *pthread_create()* function. The *fork()* function is thus used only to run new programs, and the effects of calling functions that require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()* function lets all the threads in the parent be duplicated in the child. This essentially duplicates the state of the parent in the child. This allows threads in the child to continue processing and allows locks and the state to be preserved without explicit *pthread_atfork()* code. The calling process has to ensure that the threads processing state that is shared between the parent and child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall()*. For example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*. If this is not desired behavior, the parent process has to synchronize with such threads before calling *forkall()*.

While the *fork()* function is async-signal-safe, there is no way for an implementation to determine whether the fork handlers established by *pthread_atfork()* are async-signal-safe. The fork handlers may attempt to execute portions of the implementation that are not async-signal-safe, such as those that are protected by mutexes, leading to a deadlock condition. It is therefore undefined for the fork handlers to execute functions that are not async-signal-safe when *fork()* is called from a signal handler.

When *forkall()* is called, threads, other than the calling thread, that are in functions that can return with an [EINTR] error may have those functions return [EINTR] if the implementation cannot ensure that the function behaves correctly in the parent and child. In particular, *pthread_cond_wait()* and *pthread_cond_timedwait()* need to return in order to ensure that the condition has not changed. These functions can be awakened by a spurious condition wakeup rather than returning [EINTR].

12895 FUTURE DIRECTIONS

12896 None.

12897 SEE ALSO

12898 *alarm()*, *exec*, *fcntl()*, *posix_trace_attr_getinherited()*, *posix_trace_trid_eventid_open()*,
 12899 *pthread_atfork()*, *semop()*, *signal()*, *times()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 12900 **<sys/types.h>**, **<unistd.h>**

12901 **CHANGE HISTORY**

12902 First released in Issue 1. Derived from Issue 1 of the SVID.

12903 **Issue 5**12904 The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX
12905 Threads Extension.12906 **Issue 6**12907 The following new requirements on POSIX implementations derive from alignment with the
12908 Single UNIX Specification:

- 12909 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
12910 required for conforming implementations of previous POSIX specifications, it was not
12911 required for UNIX applications.

12912 The following changes were made to align with the IEEE P1003.1a draft standard:

- 12913 • The effect of *fork()* on a pending alarm call in the child process is clarified.

12914 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

12915 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

12916 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the |
 12917 DESCRIPTION and RATIONALE relating to fork handlers registered by the *pthread_atfork()* |
 12918 function and async-signal safety. |

12919 **NAME**

12920 fpathconf, pathconf — get configurable pathname variables

12921 **SYNOPSIS**

12922 #include <unistd.h>

12923 long fpathconf(int *filde*s, int *name*);12924 long pathconf(const char **path*, int *name*);12925 **DESCRIPTION**12926 The *fpathconf()* and *pathconf()* functions shall determine the current value of a configurable limit
12927 or option (*variable*) that is associated with a file or directory.12928 For *pathconf()*, the *path* argument points to the pathname of a file or directory.12929 For *fpathconf()*, the *filde*s argument is an open file descriptor.12930 The *name* argument represents the variable to be queried relative to that file or directory.
12931 Implementations shall support all of the variables listed in the following table and may support
12932 others. The variables in the following table come from <limits.h> or <unistd.h> and the
12933 symbolic constants, defined in <unistd.h>, are the corresponding values used for *name*.

12934

12935

12936

12937

12938

12939

12940

12941

12942

12943

12944

12945

12946

12947

12948

12949

12950

12951

12952

12953

12954

Variable	Value of <i>name</i>	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	3, 4
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3, 4
{PATH_MAX}	_PC_PATH_MAX	4, 5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4, 9
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3, 4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8

12955 **Requirements**

- 12956 1. If *path* or *filde*s refers to a directory, the value returned shall apply to the directory itself.
- 12957 2. If *path* or *filde*s does not refer to a terminal file, it is unspecified whether an implementation
12958 supports an association of the variable name with the specified file.
- 12959 3. If *path* or *filde*s refers to a directory, the value returned shall apply to filenames within the
12960 directory.
- 12961 4. If *path* or *filde*s does not refer to a directory, it is unspecified whether an implementation
12962 supports an association of the variable name with the specified file.

- 12963 5. If *path* or *fildes* refers to a directory, the value returned shall be the maximum length of a
12964 relative pathname when the specified directory is the working directory.
- 12965 6. If *path* refers to a FIFO, or *fildes* refers to a pipe or FIFO, the value returned shall apply to
12966 the referenced object. If *path* or *fildes* refers to a directory, the value returned shall apply to
12967 any FIFO that exists or can be created within the directory. If *path* or *fildes* refers to any
12968 other type of file, it is unspecified whether an implementation supports an association of
12969 the variable name with the specified file.
- 12970 7. If *path* or *fildes* refers to a directory, the value returned shall apply to any files, other than
12971 directories, that exist or can be created within the directory.
- 12972 8. If *path* or *fildes* refers to a directory, it is unspecified whether an implementation supports
12973 an association of the variable name with the specified file.
- 12974 9. If *path* or *fildes* refers to a directory, the value returned shall be the maximum length of the
12975 string that a symbolic link in that directory can contain.

12976 RETURN VALUE

12977 If *name* is an invalid value, both *pathconf()* and *fpathconf()* shall return `-1` and set *errno* to
12978 indicate the error.

12979 If the variable corresponding to *name* has no limit for the *path* or file descriptor, both *pathconf()*
12980 and *fpathconf()* shall return `-1` without changing *errno*. If the implementation needs to use *path*
12981 to determine the value of *name* and the implementation does not support the association of *name*
12982 with the file specified by *path*, or if the process did not have appropriate privileges to query the
12983 file specified by *path*, or *path* does not exist, *pathconf()* shall return `-1` and set *errno* to indicate the
12984 error.

12985 If the implementation needs to use *fildes* to determine the value of *name* and the implementation
12986 does not support the association of *name* with the file specified by *fildes*, or if *fildes* is an invalid
12987 file descriptor, *fpathconf()* shall return `-1` and set *errno* to indicate the error.

12988 Otherwise, *pathconf()* or *fpathconf()* shall return the current variable value for the file or
12989 directory without changing *errno*. The value returned shall not be more restrictive than the
12990 corresponding value available to the application when it was compiled with the
12991 implementation's `<limits.h>` or `<unistd.h>`.

12992 ERRORS

12993 The *pathconf()* function shall fail if:

- 12994 [EINVAL] The value of *name* is not valid.
- 12995 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
12996 argument.
- 12997 The *pathconf()* function may fail if:
- 12998 [EACCES] Search permission is denied for a component of the path prefix.
- 12999 [EINVAL] The implementation does not support an association of the variable *name* with
13000 the specified file.
- 13001 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
13002 resolution of the *path* argument.
- 13003 [ENAMETOOLONG]
- 13004 The length of the *path* argument exceeds {PATH_MAX} or a pathname
13005 component is longer than {NAME_MAX}.

13006 [ENAMETOOLONG]
 13007 As a result of encountering a symbolic link in resolution of the *path* argument,
 13008 the length of the substituted pathname string exceeded {PATH_MAX}.

13009 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

13010 [ENOTDIR] A component of the path prefix is not a directory.

13011 The *fpathconf()* function shall fail if:

13012 [EINVAL] The value of *name* is not valid.

13013 The *fpathconf()* function may fail if:

13014 [EBADF] The *filides* argument is not a valid file descriptor.

13015 [EINVAL] The implementation does not support an association of the variable *name* with
 13016 the specified file.

13017 EXAMPLES

13018 None.

13019 APPLICATION USAGE

13020 None.

13021 RATIONALE

13022 The *pathconf()* function was proposed immediately after the *sysconf()* function when it was
 13023 realized that some configurable values may differ across file system, directory, or device
 13024 boundaries.

13025 For example, {NAME_MAX} frequently changes between System V and BSD-based file systems;
 13026 System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file
 13027 systems, an application would be forced to limit all pathname components to 14 bytes, as this
 13028 would be the value specified in <limits.h> on such a system.

13029 Therefore, various useful values can be queried on any pathname or file descriptor, assuming
 13030 that the appropriate permissions are in place.

13031 The value returned for the variable {PATH_MAX} indicates the longest relative pathname that
 13032 could be given if the specified directory is the process' current working directory. A process may
 13033 not always be able to generate a name that long and use it if a subdirectory in the pathname
 13034 crosses into a more restrictive file system.

13035 The value returned for the variable _POSIX_CHOWN_RESTRICTED also applies to directories
 13036 that do not have file systems mounted on them. The value may change when crossing a mount
 13037 point, so applications that need to know should check for each directory. (An even easier check
 13038 is to try the *chown()* function and look for an error in case it happens.)

13039 Unlike the values returned by *sysconf()*, the pathname-oriented variables are potentially more
 13040 volatile and are not guaranteed to remain constant throughout the process' lifetime. For
 13041 example, in between two calls to *pathconf()*, the file system in question may have been
 13042 unmounted and remounted with different characteristics.

13043 Also note that most of the errors are optional. If one of the variables always has the same value
 13044 on an implementation, the implementation need not look at *path* or *filides* to return that value and
 13045 is, therefore, not required to detect any of the errors except the meaning of [EINVAL] that
 13046 indicates that the value of *name* is not valid for that variable.

13047 If the value of any of the limits is unspecified (logically infinite), they will not be defined in
 13048 <limits.h> and the *pathconf()* and *fpathconf()* functions return -1 without changing *errno*. This
 13049 can be distinguished from the case of giving an unrecognized *name* argument because *errno* is set

13050 to [EINVAL] in this case.

13051 Since `-1` is a valid return value for the `pathconf()` and `fpathconf()` functions, applications should
13052 set `errno` to zero before calling them and check `errno` only if the return value is `-1`.

13053 For the case of {SYMLINK_MAX}, since both `pathconf()` and `open()` follow symbolic links, there
13054 is no way that `path` or `filenames` could refer to a symbolic link.

13055 FUTURE DIRECTIONS

13056 None.

13057 SEE ALSO

13058 `confstr()`, `sysconf()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<limits.h>`, `<unistd.h>`,
13059 the Shell and Utilities volume of IEEE Std 1003.1-2001

13060 CHANGE HISTORY

13061 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

13062 Issue 5

13063 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

13064 Large File Summit extensions are added.

13065 Issue 6

13066 The following new requirements on POSIX implementations derive from alignment with the
13067 Single UNIX Specification:

- 13068 • The DESCRIPTION is updated to include {FILESIZEBITS}.
- 13069 • The [ELOOP] mandatory error condition is added.
- 13070 • A second [ENAMETOOLONG] is added as an optional error condition.

13071 The following changes were made to align with the IEEE P1003.1a draft standard:

- 13072 • The `_PC_SYMLINK_MAX` entry is added to the table in the DESCRIPTION.

13073 The following `pathconf()` variables and their associated names are added for alignment with
13074 IEEE Std 1003.1d-1999:

13075 {POSIX_ALLOC_SIZE_MIN}
13076 {POSIX_REC_INCR_XFER_SIZE}
13077 {POSIX_REC_MAX_XFER_SIZE}
13078 {POSIX_REC_MIN_XFER_SIZE}
13079 {POSIX_REC_XFER_ALIGN}

13080 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth
13081 paragraph of the DESCRIPTION and removing shading and margin markers from the table. This
13082 change is needed since implementations are required to support all of these symbols.

13083 **NAME**

13084 fpclassify — classify real floating type

13085 **SYNOPSIS**

13086 #include <math.h>

13087 int fpclassify(real-floating x);

13088 **DESCRIPTION**

13089 cx The functionality described on this reference page is aligned with the ISO C standard. Any
13090 conflict between the requirements described here and the ISO C standard is unintentional. This
13091 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13092 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,
13093 zero, or into another implementation-defined category. First, an argument represented in a
13094 format wider than its semantic type is converted to its semantic type. Then classification is based
13095 on the type of the argument.

13096 **RETURN VALUE**

13097 The *fpclassify()* macro shall return the value of the number classification macro appropriate to
13098 the value of its argument.

13099 **ERRORS**

13100 No errors are defined.

13101 **EXAMPLES**

13102 None.

13103 **APPLICATION USAGE**

13104 None.

13105 **RATIONALE**

13106 None.

13107 **FUTURE DIRECTIONS**

13108 None.

13109 **SEE ALSO**

13110 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of
13111 IEEE Std 1003.1-2001, <math.h>

13112 **CHANGE HISTORY**

13113 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

13114 NAME

13115 fprintf, printf, snprintf, sprintf — print formatted output

13116 SYNOPSIS

13117 #include <stdio.h>

13118 int fprintf(FILE *restrict *stream*, const char *restrict *format*, ...);

13119 int printf(const char *restrict *format*, ...);

13120 int snprintf(char *restrict *s*, size_t *n*,

13121 const char *restrict *format*, ...);

13122 int sprintf(char *restrict *s*, const char *restrict *format*, ...);

13123 DESCRIPTION

13124 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13125 conflict between the requirements described here and the ISO C standard is unintentional. This
13126 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13127 The *fprintf()* function shall place output on the named output *stream*. The *printf()* function shall
13128 place output on the standard output stream *stdout*. The *sprintf()* function shall place output
13129 followed by the null byte, '\0', in consecutive bytes starting at *s*; it is the user's responsibility
13130 to ensure that enough space is available.

13131 The *snprintf()* function shall be equivalent to *sprintf()*, with the addition of the *n* argument
13132 which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s* may
13133 be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of being
13134 written to the array, and a null byte is written at the end of the bytes actually written into the
13135 array.

13136 If copying takes place between objects that overlap as a result of a call to *sprintf()* or *snprintf()*,
13137 the results are undefined.

13138 Each of these functions converts, formats, and prints its arguments under control of the *format*.
13139 The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is
13140 composed of zero or more directives: *ordinary characters*, which are simply copied to the output
13141 stream, and *conversion specifications*, each of which shall result in the fetching of zero or more
13142 arguments. The results are undefined if there are insufficient arguments for the *format*. If the
13143 *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are
13144 otherwise ignored.

13145 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
13146 to the next unused argument. In this case, the conversion specifier character % (see below) is
13147 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}],
13148 giving the position of the argument in the argument list. This feature provides for the definition
13149 of format strings that select arguments in an order appropriate to specific languages (see the
13150 EXAMPLES section).

13151 The *format* can contain either numbered argument conversion specifications (that is, "%n\$" and
13152 "*m\$"), or unnumbered argument conversion specifications (that is, % and *), but not both. The
13153 only exception to this is that %% can be mixed with the "%n\$" form. The results of mixing
13154 numbered and unnumbered argument specifications in a *format* string are undefined. When
13155 numbered argument specifications are used, specifying the *N*th argument requires that all the
13156 leading arguments, from the first to the (*N*-1)th, are specified in the format string.

13157 In format strings containing the "%n\$" form of conversion specification, numbered arguments
13158 in the argument list can be referenced from the format string as many times as required.

13159 In format strings containing the % form of conversion specification, each conversion specification
13160 uses the first unused argument in the argument list.

13161 CX	All forms of the <i>fprintf()</i> functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined in the program's locale (category <i>LC_NUMERIC</i>). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period ('.').
13162	
13163	
13164	
13165 XSI	Each conversion specification is introduced by the '%' character or by the character sequence "%n\$", after which the following appear in sequence:
13166	
13167	• Zero or more <i>flags</i> (in any order), which modify the meaning of the conversion specification.
13168	• An optional minimum <i>field width</i>. If the converted value has fewer bytes than the field width, it shall be padded with spaces by default on the left; it shall be padded on the right if the left-adjustment flag ('-'), described below, is given to the field width. The field width takes the form of an asterisk ('*'), described below, or a decimal integer.
13169	
13170	
13171	
13172	• An optional <i>precision</i> that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversion specifiers; the number of digits to appear after the radix character for the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g and G conversion specifiers; or the maximum number of bytes to be printed from a string in the s and S conversion specifiers. The precision takes the form of a period ('.') followed either by an asterisk ('*'), described below, or an optional decimal digit string, where a null digit string is treated as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
13173	
13174	
13175	
13176 XSI	
13177	
13178	
13179	
13180	• An optional length modifier that specifies the size of the argument.
13181	• A <i>conversion specifier</i> character that indicates the type of conversion to be applied.
13182	A field width, or precision, or both, may be indicated by an asterisk ('*'). In this case an argument of type int supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format strings containing the "%n\$" form of a conversion specification, a field width or precision may be indicated by the sequence "*m\$", where <i>m</i> is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the <i>format</i> argument) of an integer argument containing the field width or precision, for example:
13183	
13184	
13185	
13186 XSI	
13187	
13188	
13189	
13190	
13191	<pre>printf("%1\$d:%2\$.*3\$d:%4\$.*3\$d\n", hour, min, precision, sec);</pre>
13192	The flag characters and their meanings are:
13193 XSI	
13194	
13195	' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.
13196	- The result of the conversion shall be left-justified within the field. The conversion is right-justified if this flag is not specified.
13197	
13198	+ The result of a signed conversion shall always begin with a sign ('+' or '-'). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.
13199	
13200	
13201	<space> If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.
13202	
13203	
13204	# Specifies that the value is to be converted to an alternative form. For o conversion, it increases the precision (if necessary) to force the first digit of the result to be zero. For x
13205	

13206 or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A,
 13207 e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix
 13208 character, even if no digits follow the radix character. Without this flag, a radix
 13209 character appears in the result of these conversions only if a digit follows it. For g and G
 13210 conversion specifiers, trailing zeros shall *not* be removed from the result as they
 13211 normally are. For other conversion specifiers, the behavior is undefined.

13212 0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros
 13213 (following any indication of sign or base) are used to pad to the field width; no space
 13214 padding is performed. If the '0' and '-' flags both appear, the '0' flag is ignored. For
 13215 d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag is
 13216 xsl ignored. If the '0' and ' ' flags both appear, the grouping characters are inserted
 13217 before zero padding. For other conversions, the behavior is undefined.

13218 The length modifiers and their meanings are:

13219 hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char**
 13220 or **unsigned char** argument (the argument will have been promoted according to the
 13221 integer promotions, but its value shall be converted to **signed char** or **unsigned char**
 13222 before printing); or that a following n conversion specifier applies to a pointer to a
 13223 **signed char** argument.

13224 h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or
 13225 **unsigned short** argument (the argument will have been promoted according to the
 13226 integer promotions, but its value shall be converted to **short** or **unsigned short** before
 13227 printing); or that a following n conversion specifier applies to a pointer to a **short**
 13228 argument.

13229 l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or
 13230 **unsigned long** argument; that a following n conversion specifier applies to a pointer to
 13231 a **long** argument; that a following c conversion specifier applies to a **wint_t** argument;
 13232 that a following s conversion specifier applies to a pointer to a **wchar_t** argument; or
 13233 has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.

13234 ll (ell-ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long** or
 13235 **unsigned long long** argument; or that a following n conversion specifier applies to a
 13236 pointer to a **long long** argument.

13238 j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax_t**
 13239 or **uintmax_t** argument; or that a following n conversion specifier applies to a pointer
 13240 to an **intmax_t** argument.

13241 z Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **size_t** or the
 13242 corresponding signed integer type argument; or that a following n conversion specifier
 13243 applies to a pointer to a signed integer type corresponding to a **size_t** argument.

13244 t Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **ptrdiff_t** or
 13245 the corresponding **unsigned** type argument; or that a following n conversion specifier
 13246 applies to a pointer to a **ptrdiff_t** argument.

13247 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a **long**
 13248 **double** argument.

13249 If a length modifier appears with any conversion specifier other than as specified above, the
 13250 behavior is undefined.

13251		The conversion specifiers and their meanings are:
13252	d, i	The int argument shall be converted to a signed decimal in the style "[-] dddd". The
13253		precision specifies the minimum number of digits to appear; if the value being
13254		converted can be represented in fewer digits, it shall be expanded with leading zeros.
13255		The default precision is 1. The result of converting zero with an explicit precision of
13256		zero shall be no characters.
13257	o	The unsigned argument shall be converted to unsigned octal format in the style
13258		"ddd". The precision specifies the minimum number of digits to appear; if the value
13259		being converted can be represented in fewer digits, it shall be expanded with leading
13260		zeros. The default precision is 1. The result of converting zero with an explicit precision
13261		of zero shall be no characters.
13262	u	The unsigned argument shall be converted to unsigned decimal format in the style
13263		"ddd". The precision specifies the minimum number of digits to appear; if the value
13264		being converted can be represented in fewer digits, it shall be expanded with leading
13265		zeros. The default precision is 1. The result of converting zero with an explicit precision
13266		of zero shall be no characters.
13267	x	The unsigned argument shall be converted to unsigned hexadecimal format in the style
13268		"ddd"; the letters "abcdef" are used. The precision specifies the minimum number
13269		of digits to appear; if the value being converted can be represented in fewer digits, it
13270		shall be expanded with leading zeros. The default precision is 1. The result of
13271		converting zero with an explicit precision of zero shall be no characters.
13272	X	Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead
13273		of "abcdef".
13274	f, F	The double argument shall be converted to decimal notation in the style
13275		"[-] ddd.ddd", where the number of digits after the radix character is equal to the
13276		precision specification. If the precision is missing, it shall be taken as 6; if the precision
13277		is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix
13278		character appears, at least one digit appears before it. The low-order digit shall be
13279		rounded in an implementation-defined manner.
13280		A double argument representing an infinity shall be converted in one of the styles
13281		"[-] inf" or "[-] infinity"; which style is implementation-defined. A double
13282		argument representing a NaN shall be converted in one of the styles "[-] nan(<i>n-char-sequence</i>)
13283		" or "[-] nan"; which style, and the meaning of any <i>n-char-sequence</i> ,
13284		is implementation-defined. The F conversion specifier produces "INF", "INFINITY",
13285		or "NAN" instead of "inf", "infinity", or "nan", respectively.
13286	e, E	The double argument shall be converted in the style "[-] d.ddde±dd", where there is
13287		one digit before the radix character (which is non-zero if the argument is non-zero) and
13288		the number of digits after it is equal to the precision; if the precision is missing, it shall
13289		be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall
13290		appear. The low-order digit shall be rounded in an implementation-defined manner.
13291		The E conversion specifier shall produce a number with 'E' instead of 'e'
13292		introducing the exponent. The exponent shall always contain at least two digits. If the
13293		value is zero, the exponent shall be zero.
13294		A double argument representing an infinity or NaN shall be converted in the style of
13295		an f or F conversion specifier.
13296	g, G	The double argument shall be converted in the style f or e (or in the style F or E in the
13297		case of a G conversion specifier), with the precision specifying the number of significant

13298		digits. If an explicit precision is zero, it shall be taken as 1. The style used depends on
13299		the value converted; style <code>e</code> (or <code>E</code>) shall be used only if the exponent resulting from
13300		such a conversion is less than <code>-4</code> or greater than or equal to the precision. Trailing zeros
13301		shall be removed from the fractional portion of the result; a radix character shall appear
13302		only if it is followed by a digit or a <code>'#'</code> flag is present.
13303		A double argument representing an infinity or NaN shall be converted in the style of
13304		an <code>f</code> or <code>F</code> conversion specifier.
13305	a, A	A double argument representing a floating-point number shall be converted in the
13306		style <code>"[-] 0xh.hhhhp±d"</code> , where there is one hexadecimal digit (which shall be non-
13307		zero if the argument is a normalized floating-point number and is otherwise
13308		unspecified) before the decimal-point character and the number of hexadecimal digits
13309		after it is equal to the precision; if the precision is missing and <code>FLT_RADIX</code> is a power
13310		of 2, then the precision shall be sufficient for an exact representation of the value; if the
13311		precision is missing and <code>FLT_RADIX</code> is not a power of 2, then the precision shall be
13312		sufficient to distinguish values of type double , except that trailing zeros may be
13313		omitted; if the precision is zero and the <code>'#'</code> flag is not specified, no decimal-point
13314		character shall appear. The letters <code>"abcdef"</code> shall be used for a conversion and the
13315		letters <code>"ABCDEF"</code> for <code>A</code> conversion. The <code>A</code> conversion specifier produces a number with
13316		<code>'X'</code> and <code>'P'</code> instead of <code>'x'</code> and <code>'p'</code> . The exponent shall always contain at least one
13317		digit, and only as many more digits as necessary to represent the decimal exponent of
13318		2. If the value is zero, the exponent shall be zero.
13319		A double argument representing an infinity or NaN shall be converted in the style of
13320		an <code>f</code> or <code>F</code> conversion specifier.
13321	c	The int argument shall be converted to an unsigned char , and the resulting byte shall
13322		be written.
13323		If an <code>l</code> (ell) qualifier is present, the wint_t argument shall be converted as if by an <code>ls</code>
13324		conversion specification with no precision and an argument that points to a two-
13325		element array of type wchar_t , the first element of which contains the wint_t argument
13326		to the <code>ls</code> conversion specification and the second element contains a null wide
13327		character.
13328	s	The argument shall be a pointer to an array of char . Bytes from the array shall be
13329		written up to (but not including) any terminating null byte. If the precision is specified,
13330		no more than that many bytes shall be written. If the precision is not specified or is
13331		greater than the size of the array, the application shall ensure that the array contains a
13332		null byte.
13333		If an <code>l</code> (ell) qualifier is present, the argument shall be a pointer to an array of type
13334		wchar_t . Wide characters from the array shall be converted to characters (each as if by
13335		a call to the <code>wcrtomb()</code> function, with the conversion state described by an mbstate_t
13336		object initialized to zero before the first wide character is converted) up to and
13337		including a terminating null wide character. The resulting characters shall be written
13338		up to (but not including) the terminating null character (byte). If no precision is
13339		specified, the application shall ensure that the array contains a null wide character. If a
13340		precision is specified, no more than that many characters (bytes) shall be written
13341		(including shift sequences, if any), and the array shall contain a null wide character if,
13342		to equal the character sequence length given by the precision, the function would need
13343		to access a wide character one past the end of the array. In no case shall a partial
13344		character be written.

13345	p	The argument shall be a pointer to void . The value of the pointer is converted to a
13346		sequence of printable characters, in an implementation-defined manner.
13347	n	The argument shall be a pointer to an integer into which is written the number of bytes
13348		written to the output so far by this call to one of the <i>fprintf()</i> functions. No argument is
13349		converted.
13350	XSI C	Equivalent to <code>lc</code> .
13351	XSI S	Equivalent to <code>ls</code> .
13352	%	Print a '%' character; no argument is converted. The complete conversion specification
13353		shall be <code>%%</code> .
13354		If a conversion specification does not match one of the above forms, the behavior is undefined. If
13355		any argument is not the correct type for the corresponding conversion specification, the
13356		behavior is undefined.
13357		In no case shall a nonexistent or small field width cause truncation of a field; if the result of a
13358		conversion is wider than the field width, the field shall be expanded to contain the conversion
13359		result. Characters generated by <i>fprintf()</i> and <i>printf()</i> are printed as if <i>fputc()</i> had been called.
13360		For the <code>a</code> and <code>A</code> conversion specifiers, if <code>FLT_RADIX</code> is a power of 2, the value shall be correctly
13361		rounded to a hexadecimal floating number with the given precision.
13362		For <code>a</code> and <code>A</code> conversions, if <code>FLT_RADIX</code> is not a power of 2 and the result is not exactly
13363		representable in the given precision, the result should be one of the two adjacent numbers in
13364		hexadecimal floating style with the given precision, with the extra stipulation that the error
13365		should have a correct sign for the current rounding direction.
13366		For the <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , and <code>G</code> conversion specifiers, if the number of significant decimal digits is at
13367		most <code>DECIMAL_DIG</code> , then the result should be correctly rounded. If the number of significant
13368		decimal digits is more than <code>DECIMAL_DIG</code> but the source value is exactly representable with
13369		<code>DECIMAL_DIG</code> digits, then the result should be an exact representation with trailing zeros.
13370		Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having
13371		<code>DECIMAL_DIG</code> significant digits; the value of the resultant decimal string D should satisfy $L \leq$
13372		$D \leq U$, with the extra stipulation that the error should have a correct sign for the current
13373		rounding direction.
13374	CX	The <code>st_ctime</code> and <code>st_mtime</code> fields of the file shall be marked for update between the call to a
13375		successful execution of <i>fprintf()</i> or <i>printf()</i> and the next successful completion of a call to <i>fflush()</i>
13376		or <i>fclose()</i> on the same stream or a call to <i>exit()</i> or <i>abort()</i> .
13377	RETURN VALUE	
13378		Upon successful completion, the <i>fprintf()</i> and <i>printf()</i> functions shall return the number of bytes
13379		transmitted.
13380		Upon successful completion, the <i>sprintf()</i> function shall return the number of bytes written to <code>s</code> ,
13381		excluding the terminating null byte.
13382		Upon successful completion, the <i>snprintf()</i> function shall return the number of bytes that would
13383		be written to <code>s</code> had <code>n</code> been sufficiently large excluding the terminating null byte.
13384		If an output error was encountered, these functions shall return a negative value.
13385		If the value of <code>n</code> is zero on a call to <i>snprintf()</i> , nothing shall be written, the number of bytes that
13386		would have been written had <code>n</code> been sufficiently large excluding the terminating null shall be
13387		returned, and <code>s</code> may be a null pointer.

13388 ERRORS

13389 For the conditions under which *fprintf()* and *printf()* fail and may fail, refer to *fputc()* or
13390 *fputwc()*.

13391 In addition, all forms of *fprintf()* may fail if:

13392 XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been
13393 detected.

13394 XSI [EINVAL] There are insufficient arguments.

13395 The *printf()* and *fprintf()* functions may fail if:

13396 XSI [ENOMEM] Insufficient storage space is available.

13397 The *snprintf()* function shall fail if:

13398 XSI [EOVERFLOW] The value of *n* is greater than {INT_MAX} or the number of bytes needed to
13399 hold the output excluding the terminating null is greater than {INT_MAX}.

13400 EXAMPLES

13401 Printing Language-Independent Date and Time

13402 The following statement can be used to print date and time using a language-independent
13403 format:

13404 `printf(format, weekday, month, day, hour, min);`

13405 For American usage, *format* could be a pointer to the following string:

13406 `"%s, %s %d, %d:%.2d\n"`

13407 This example would produce the following message:

13408 Sunday, July 3, 10:02

13409 For German usage, *format* could be a pointer to the following string:

13410 `"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"`

13411 This definition of *format* would produce the following message:

13412 Sonntag, 3. Juli, 10:02

13413 Printing File Information

13414 The following example prints information about the type, permissions, and number of links of a
13415 specific file in a directory.

13416 The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined
13417 *strperm()* function shall return a string similar to the one at the beginning of the output for the
13418 following command:

13419 `ls -l`

13420 The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()*
13421 function shall return a **passwd** structure from which the name of the user is extracted. If the user
13422 name is not found, the program instead prints out the numeric value of the user ID.

13423 The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar to
13424 *getpwuid()* except that it shall return group information based on the group number. Once
13425 again, if the group is not found, the program prints the numeric value of the group for the entry.


```

13426     The final call to printf() prints the size of the file.
13427     #include <stdio.h>
13428     #include <sys/types.h>
13429     #include <pwd.h>
13430     #include <grp.h>
13431
13432     char *strperm (mode_t);
13433     ...
13434     struct stat statbuf;
13435     struct passwd *pwd;
13436     struct group *grp;
13437     ...
13438     printf("%10.10s", strperm (statbuf.st_mode));
13439     printf("%4d", statbuf.st_nlink);
13440
13441     if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
13442         printf(" %-8.8s", pwd->pw_name);
13443     else
13444         printf(" %-8ld", (long) statbuf.st_uid);
13445
13446     if ((grp = getgrgid(statbuf.st_gid)) != NULL)
13447         printf(" %-8.8s", grp->gr_name);
13448     else
13449         printf(" %-8ld", (long) statbuf.st_gid);
13450
13451     printf("%9jd", (intmax_t) statbuf.st_size);
13452     ...

```

Printing a Localized Date String

```

13453     The following example gets a localized date string. The nl_langinfo() function shall return the
13454     localized date string, which specifies the order and layout of the date. The strftime() function
13455     takes this information and, using the tm structure for values, places the date and time
13456     information into datestring. The printf() function then outputs datestring and the name of the
13457     entry.
13458
13459     #include <stdio.h>
13460     #include <time.h>
13461     #include <langinfo.h>
13462     ...
13463     struct dirent *dp;
13464     struct tm *tm;
13465     char datestring[256];
13466     ...
13467     strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
13468
13469     printf(" %s %s\n", datestring, dp->d_name);
13470     ...

```

Printing Error Information

The following example uses *fprintf()* to write error information to standard error.

In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If the file already exists, this is an error, as indicated by the **O_EXCL** flag on the *open()* function. If the call fails, the program assumes that someone else is updating the password file, and the program exits.

The next group of calls saves a new password file as the current password file by creating a link between **LOCKFILE** and the new password file **PASSWDFILE**.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
...
int pfd;
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
if (link(LOCKFILE, PASSWDFILE) == -1) {
    fprintf(stderr, "Link error: %s\n", strerror(errno));
    exit(1);
}
...
```

Printing Usage Information

The following example checks to make sure the program has the necessary arguments, and uses *fprintf()* to print usage information if the expected number of arguments is not present.

```
#include <stdio.h>
#include <stdlib.h>
...
char *Options = "hdbtl";
...
if (argc < 2) {
    fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
}
...
```

Formatting a Decimal String

The following example prints a key and data pair on *stdout*. Note use of the '*' (asterisk) in the format string; this ensures the correct number of decimal places for the element based on the number of elements requested.

```
#include <stdio.h>
...
long i;
char *keyst;
int elementlen, len;
...
while (len < elementlen) {
...
    printf("%s Element%0*ld\n", keyst, elementlen, i);
...
}
```

Creating a Filename

The following example creates a filename using information from a previous *getpwnam()* function that returned the HOME directory of the user.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
...
char filename[PATH_MAX+1];
struct passwd *pw;
...
sprintf(filename, "%s/%d.out", pw->pw_dir, getpid());
...
```

Reporting an Event

The following example loops until an event has timed out. The *pause()* function waits forever unless it receives a signal. The *fprintf()* statement should never occur due to the possible return values of *pause()*.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
...
while (!event_complete) {
...
    if (pause() != -1 || errno != EINTR)
        fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
...
}
```

Printing Monetary Information

The following example uses *strfmon()* to convert a number and store it as a formatted monetary string named *convbuf*. If the first number is printed, the program prints the format and the description; otherwise, it just prints the number.

```
#include <monetary.h>
#include <stdio.h>
...
struct tblfmt {
    char *format;
    char *description;
};

struct tblfmt table[] = {
    { "%n", "default formatting" },
    { "%11n", "right align within an 11 character field" },
    { "%#5n", "aligned columns for values up to 99999" },
    { "%=*#5n", "specify a fill character" },
    { "%=0#5n", "fill characters do not use grouping" },
    { "%^#5n", "disable the grouping separator" },
    { "%^#5.0n", "round off to whole units" },
    { "%^#5.4n", "increase the precision" },
    { "%(#5n", "use an alternative pos/neg style" },
    { "%!(#5n", "disable the currency symbol" },
};
...
float input[3];
int i, j;
char convbuf[100];
...
strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);

if (j == 0) {
    printf("%s%s\n", table[i].format,
           convbuf, table[i].description);
}
else {
    printf("%s\n", convbuf);
}
...
```

Printing Wide Characters

The following example prints a series of wide characters. Suppose that "L'@'" expands to three bytes:

```
wchar_t wz [3] = L"@@";           // Zero-terminated
wchar_t wn [3] = L"@@";           // Unterminated

fprintf (stdout, "%ls", wz);       // Outputs 6 bytes
fprintf (stdout, "%ls", wn);       // Undefined because wn has no terminator
fprintf (stdout, "%4ls", wz);      // Outputs 3 bytes
fprintf (stdout, "%4ls", wn);      // Outputs 3 bytes; no terminator needed
fprintf (stdout, "%9ls", wz);      // Outputs 6 bytes
```

```

13600 fprintf (stdout,"%9ls", wn); // Outputs 9 bytes; no terminator needed
13601 fprintf (stdout,"%10ls", wz); // Outputs 6 bytes
13602 fprintf (stdout,"%10ls", wn); // Undefined because wn has no terminator

```

13603 In the last line of the example, after processing three characters, nine bytes have been output.
 13604 The fourth character must then be examined to determine whether it converts to one byte or
 13605 more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth
 13606 character in the array, the behavior is undefined.

13607 APPLICATION USAGE

13608 If the application calling *fprintf()* has any objects of type **wint_t** or **wchar_t**, it must also include
 13609 the **<wchar.h>** header to have these objects defined.

13610 RATIONALE

13611 None.

13612 FUTURE DIRECTIONS

13613 None.

13614 SEE ALSO

13615 *fputc()*, *fscanf()*, *setlocale()*, *strfmon()*, *wcrtomb()*, the Base Definitions volume of
 13616 IEEE Std 1003.1-2001, Chapter 7, Locale, **<stdio.h>**, **<wchar.h>**

13617 CHANGE HISTORY

13618 First released in Issue 1. Derived from Issue 1 of the SVID.

13619 Issue 5

13620 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the *l* (ell) qualifier can
 13621 now be used with *c* and *s* conversion specifiers.

13622 The *snprintf()* function is new in Issue 5.

13623 Issue 6

13624 Extensions beyond the ISO C standard are marked.

13625 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

13626 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 13627 • The prototypes for *fprintf()*, *printf()*, *snprintf()*, and *sprintf()* are updated, and the XSI
- 13628 shading is removed from *snprintf()*.
- 13629 • The description of *snprintf()* is aligned with the ISO C standard. Note that this supersedes
- 13630 the *snprintf()* description in The Open Group Base Resolution bwg98-006, which changed the
- 13631 behavior from Issue 5.
- 13632 • The DESCRIPTION is updated.

13633 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
 13634 specification” consistently.

13635 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

13636 An example of printing wide characters is added.

13637 **NAME**

13638 fputc — put a byte on a stream

13639 **SYNOPSIS**

13640 #include <stdio.h>

13641 int fputc(int *c*, FILE **stream*);13642 **DESCRIPTION**

13643 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13644 conflict between the requirements described here and the ISO C standard is unintentional. This
 13645 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13646 The *fputc()* function shall write the byte specified by *c* (converted to an **unsigned char**) to the
 13647 output stream pointed to by *stream*, at the position indicated by the associated file-position
 13648 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file
 13649 cannot support positioning requests, or if the stream was opened with append mode, the byte
 13650 shall be appended to the output stream.

13651 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
 13652 execution of *fputc()* and the next successful completion of a call to *fflush()* or *fclose()* on the same
 13653 stream or a call to *exit()* or *abort()*.

13654 **RETURN VALUE**

13655 Upon successful completion, *fputc()* shall return the value it has written. Otherwise, it shall
 13656 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the
 13657 error.

13658 **ERRORS**

13659 The *fputc()* function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be
 13660 flushed, and:

13661 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 13662 process would be delayed in the write operation.

13663 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 13664 writing.

13665 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

13666 XSI [EFBIG] An attempt was made to write to a file that exceeds the process' file size limit.

13667 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 13668 offset maximum.

13669 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 13670 was transferred.

13671 CX [EIO] A physical I/O error has occurred, or the process is a member of a
 13672 background process group attempting to write to its controlling terminal,
 13673 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the
 13674 process group of the process is orphaned. This error may also be returned
 13675 under implementation-defined conditions.

13676 CX [ENOSPC] There was no free space remaining on the device containing the file.

13677 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 13678 any process. A SIGPIPE signal shall also be sent to the thread.

13679 The *fputc()* function may fail if:

13680 CX [ENOMEM] Insufficient storage space is available.

13681 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
13682 capabilities of the device.

13683 **EXAMPLES**

13684 None.

13685 **APPLICATION USAGE**

13686 None.

13687 **RATIONALE**

13688 None.

13689 **FUTURE DIRECTIONS**

13690 None.

13691 **SEE ALSO**

13692 *ferror()*, *fopen()*, *getrlimit()*, *putc()*, *puts()*, *setbuf()*, *ulimit()*, the Base Definitions volume of
13693 IEEE Std 1003.1-2001, **<stdio.h>**

13694 **CHANGE HISTORY**

13695 First released in Issue 1. Derived from Issue 1 of the SVID.

13696 **Issue 5**

13697 Large File Summit extensions are added.

13698 **Issue 6**

13699 Extensions beyond the ISO C standard are marked.

13700 The following new requirements on POSIX implementations derive from alignment with the
13701 Single UNIX Specification:

- 13702 • The [EIO] and [EFBIG] mandatory error conditions are added.
- 13703 • The [ENOMEM] and [ENXIO] optional error conditions are added.

13704 **NAME**

13705 fputs — put a string on a stream

13706 **SYNOPSIS**

13707 #include <stdio.h>

13708 int fputs(const char *restrict *s*, FILE *restrict *stream*);13709 **DESCRIPTION**

13710 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13711 conflict between the requirements described here and the ISO C standard is unintentional. This
 13712 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13713 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed
 13714 to by *stream*. The terminating null byte shall not be written.

13715 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
 13716 execution of *fputs()* and the next successful completion of a call to *fflush()* or *fclose()* on the same
 13717 stream or a call to *exit()* or *abort()*.

13718 **RETURN VALUE**

13719 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall
 13720 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

13721 **ERRORS**13722 Refer to *fputc()*.13723 **EXAMPLES**13724 **Printing to Standard Output**

13725 The following example gets the current time, converts it to a string using *localtime()* and
 13726 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to
 13727 an event for which it is waiting.

```
13728 #include <time.h>
13729 #include <stdio.h>
13730 ...
13731 time_t now;
13732 int minutes_to_event;
13733 ...
13734 time(&now);
13735 printf("The time is ");
13736 fputs(asctime(localtime(&now)), stdout);
13737 printf("There are still %d minutes to the event.\n",
13738       minutes_to_event);
13739 ...
```

13740 **APPLICATION USAGE**13741 The *puts()* function appends a <newline> while *fputs()* does not.13742 **RATIONALE**

13743 None.

13744 **FUTURE DIRECTIONS**

13745 None.

13746 **SEE ALSO**

13747 *fopen()*, *putc()*, *puts()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

13748 **CHANGE HISTORY**

13749 First released in Issue 1. Derived from Issue 1 of the SVID.

13750 **Issue 6**

13751 Extensions beyond the ISO C standard are marked.

13752 The *fputs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

13753 **NAME**

13754 fputc — put a wide-character code on a stream

13755 **SYNOPSIS**

13756 #include <stdio.h>

13757 #include <wchar.h>

13758 wint_t fputc(wchar_t wc, FILE *stream);

13759 **DESCRIPTION**

13760 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13761 conflict between the requirements described here and the ISO C standard is unintentional. This
 13762 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13763 The *fputc()* function shall write the character corresponding to the wide-character code *wc* to
 13764 the output stream pointed to by *stream*, at the position indicated by the associated file-position
 13765 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot
 13766 support positioning requests, or if the stream was opened with append mode, the character is
 13767 appended to the output stream. If an error occurs while writing the character, the shift state of
 13768 the output file is left in an undefined state.

13769 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
 13770 execution of *fputc()* and the next successful completion of a call to *fflush()* or *fclose()* on the
 13771 same stream or a call to *exit()* or *abort()*.

13772 **RETURN VALUE**

13773 Upon successful completion, *fputc()* shall return *wc*. Otherwise, it shall return WEOF, the error
 13774 CX indicator for the stream shall be set, and *errno* shall be set to indicate the error.

13775 **ERRORS**

13776 The *fputc()* function shall fail if either the stream is unbuffered or data in the *stream*'s buffer
 13777 needs to be written, and:

13778 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 13779 process would be delayed in the write operation.

13780 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 13781 writing.

13782 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size or
 13783 the process' file size limit.

13784 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 13785 offset maximum associated with the corresponding stream.

13786 [EILSEQ] The wide-character code *wc* does not correspond to a valid character.

13787 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 13788 was transferred.

13789 CX [EIO] A physical I/O error has occurred, or the process is a member of a
 13790 background process group attempting to write to its controlling terminal,
 13791 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the
 13792 process group of the process is orphaned. This error may also be returned
 13793 under implementation-defined conditions.

13794 CX [ENOSPC] There was no free space remaining on the device containing the file.

13795 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 13796 any process. A SIGPIPE signal shall also be sent to the thread.

13797 The *fputwc()* function may fail if:

13798 CX [ENOMEM] Insufficient storage space is available.

13799 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
13800 capabilities of the device.

13801 EXAMPLES

13802 None.

13803 APPLICATION USAGE

13804 None.

13805 RATIONALE

13806 None.

13807 FUTURE DIRECTIONS

13808 None.

13809 SEE ALSO

13810 *ferror()*, *fopen()*, *setbuf()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-2001,
13811 **<stdio.h>**, **<wchar.h>**

13812 CHANGE HISTORY

13813 First released in Issue 4. Derived from the MSE working draft.

13814 Issue 5

13815 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
13816 is changed from **wint_t** to **wchar_t**.

13817 The Optional Header (OH) marking is removed from **<stdio.h>**.

13818 Large File Summit extensions are added.

13819 Issue 6

13820 Extensions beyond the ISO C standard are marked.

13821 The following new requirements on POSIX implementations derive from alignment with the
13822 Single UNIX Specification:

- 13823 • The [EFBIG] and [EIO] mandatory error conditions are added.
- 13824 • The [ENOMEM] and [ENXIO] optional error conditions are added.

13825 **NAME**

13826 fputws — put a wide-character string on a stream

13827 **SYNOPSIS**

13828 #include <stdio.h>

13829 #include <wchar.h>

13830 int fputws(const wchar_t *restrict ws, FILE *restrict stream);

13831 **DESCRIPTION**

13832 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13833 conflict between the requirements described here and the ISO C standard is unintentional. This
 13834 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13835 The *fputws()* function shall write a character string corresponding to the (null-terminated)
 13836 wide-character string pointed to by *ws* to the stream pointed to by *stream*. No character
 13837 corresponding to the terminating null wide-character code shall be written.

13838 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
 13839 execution of *fputws()* and the next successful completion of a call to *fflush()* or *fclose()* on the
 13840 same stream or a call to *exit()* or *abort()*.

13841 **RETURN VALUE**

13842 Upon successful completion, *fputws()* shall return a non-negative number. Otherwise, it shall
 13843 CX return -1, set an error indicator for the stream, and set *errno* to indicate the error.

13844 **ERRORS**13845 Refer to *fputwc()*.13846 **EXAMPLES**

13847 None.

13848 **APPLICATION USAGE**13849 The *fputws()* function does not append a <newline>.13850 **RATIONALE**

13851 None.

13852 **FUTURE DIRECTIONS**

13853 None.

13854 **SEE ALSO**13855 *fopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>, <wchar.h>13856 **CHANGE HISTORY**

13857 First released in Issue 4. Derived from the MSE working draft.

13858 **Issue 5**

13859 The Optional Header (OH) marking is removed from <stdio.h>.

13860 **Issue 6**

13861 Extensions beyond the ISO C standard are marked.

13862 The *fputws()* prototype is updated for alignment with the ISO / IEC 9899:1999 standard.

13863 **NAME**

13864 fread — binary input

13865 **SYNOPSIS**

13866 #include <stdio.h>

```
13867     size_t fread(void *restrict ptr, size_t size, size_t nitems,
13868                 FILE *restrict stream);
```

13869 **DESCRIPTION**

13870 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13871 conflict between the requirements described here and the ISO C standard is unintentional. This
 13872 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13873 The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* elements whose size
 13874 is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall
 13875 be made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned**
 13876 **char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be
 13877 advanced by the number of bytes successfully read. If an error occurs, the resulting value of the
 13878 file position indicator for the stream is unspecified. If a partial element is read, its value is
 13879 unspecified.

13880 CX The *fread()* function may mark the *st_atime* field of the file associated with *stream* for update. The
 13881 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 13882 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 13883 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

13884 **RETURN VALUE**

13885 Upon successful completion, *fread()* shall return the number of elements successfully read which
 13886 is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread()*
 13887 shall return 0 and the contents of the array and the state of the stream remain unchanged.
 13888 CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall be
 13889 set to indicate the error.

13890 **ERRORS**13891 Refer to *fgetc()*.13892 **EXAMPLES**13893 **Reading from a Stream**13894 The following example reads a single element from the *fp* stream into the array pointed to by *buf*.

```
13895 #include <stdio.h>
13896 ...
13897 size_t bytes_read;
13898 char buf[100];
13899 FILE *fp;
13900 ...
13901 bytes_read = fread(buf, sizeof(buf), 1, fp);
13902 ...
```

13903 **APPLICATION USAGE**

13904 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
 13905 end-of-file condition.

13906 Because of possible differences in element length and byte ordering, files written using *fwrite()*
 13907 are application-dependent, and possibly cannot be read using *fread()* by a different application

13908 or by the same application on a different processor.

13909 **RATIONALE**

13910 None.

13911 **FUTURE DIRECTIONS**

13912 None.

13913 **SEE ALSO**

13914 *feof()*, *ferror()*, *fgetc()*, *fopen()*, *getc()*, *gets()*, *scanf()*, the Base Definitions volume of
13915 IEEE Std 1003.1-2001, **<stdio.h>**

13916 **CHANGE HISTORY**

13917 First released in Issue 1. Derived from Issue 1 of the SVID.

13918 **Issue 6**

13919 Extensions beyond the ISO C standard are marked.

13920 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 13921 • The *fread()* prototype is updated.
- 13922 • The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

13923 **NAME**

13924 free — free allocated memory

13925 **SYNOPSIS**

13926 #include <stdlib.h>

13927 void free(void *ptr);

13928 **DESCRIPTION**

13929 CX The functionality described on this reference page is aligned with the ISO C standard. Any
13930 conflict between the requirements described here and the ISO C standard is unintentional. This
13931 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13932 The *free()* function shall cause the space pointed to by *ptr* to be deallocated; that is, made
13933 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the
13934 ADV argument does not match a pointer earlier returned by the *calloc()*, *malloc()*, *posix_memalign()*,
13935 XSI *realloc()*, or *strdup()* function, or if the space has been deallocated by a call to *free()* or *realloc()*,
13936 the behavior is undefined.

13937 Any use of a pointer that refers to freed space results in undefined behavior.

13938 **RETURN VALUE**13939 The *free()* function shall not return a value.13940 **ERRORS**

13941 No errors are defined.

13942 **EXAMPLES**

13943 None.

13944 **APPLICATION USAGE**

13945 There is now no requirement for the implementation to support the inclusion of <malloc.h>.

13946 **RATIONALE**

13947 None.

13948 **FUTURE DIRECTIONS**

13949 None.

13950 **SEE ALSO**13951 *calloc()*, *malloc()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>13952 **CHANGE HISTORY**

13953 First released in Issue 1. Derived from Issue 1 of the SVID.

13954 **Issue 6**13955 Reference to the *valloc()* function is removed.

13956 **NAME**

13957 freeaddrinfo, getaddrinfo — get address information

13958 **SYNOPSIS**

```

13959     #include <sys/socket.h>
13960     #include <netdb.h>

13961     void freeaddrinfo(struct addrinfo *ai);
13962     int getaddrinfo(const char *restrict nodename,
13963                    const char *restrict servname,
13964                    const struct addrinfo *restrict hints,
13965                    struct addrinfo **restrict res);

```

13966 **DESCRIPTION**

13967 The *freeaddrinfo()* function shall free one or more **addrinfo** structures returned by *getaddrinfo()*,
 13968 along with any additional storage associated with those structures. If the *ai_next* field of the
 13969 structure is not null, the entire list of structures shall be freed. The *freeaddrinfo()* function shall
 13970 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo()*.

13971 The *getaddrinfo()* function shall translate the name of a service location (for example, a host
 13972 name) and/or a service name and shall return a set of socket addresses and associated
 13973 information to be used in creating a socket with which to address the specified service.

13974 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,
 13975 RFC 1035, and RFC 1886.

13976 The *freeaddrinfo()* and *getaddrinfo()* functions shall be thread-safe.

13977 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated
 13978 strings. One or both of these two arguments shall be supplied by the application as a non-null
 13979 pointer.

13980 The format of a valid name depends on the address family or families. If a specific family is not
 13981 given and the name could be interpreted as valid within multiple supported families, the
 13982 implementation shall attempt to resolve the name in all supported families and, in absence of
 13983 errors, one or more results shall be returned.

13984 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If
 13985 IP6 the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, valid descriptive names
 13986 include host names. If the specified address family is AF_INET or AF_UNSPEC, address strings
 13987 using Internet standard dot notation as specified in *inet_addr()* are valid.

13988 IP6 If the specified address family is AF_INET6 or AF_UNSPEC, standard IPv6 text forms described
 13989 in *inet_ntop()* are valid.

13990 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the
 13991 requested service location is local to the caller.

13992 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If
 13993 *servname* is not null, it is a null-terminated character string identifying the requested service. This
 13994 can be either a descriptive name or a numeric representation suitable for use with the address
 13995 IP6 family or families. If the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, the
 13996 service can be specified as a string specifying a decimal port number.

13997 If the *hints* argument is not null, it refers to a structure containing input values that may direct
 13998 the operation by providing options and by limiting the returned information to a specific socket
 13999 type, address family, and/or protocol. In this *hints* structure every member other than *ai_flags*,
 14000 *ai_family*, *ai_socktype*, and *ai_protocol* shall be set to zero or a null pointer. A value of
 14001 AF_UNSPEC for *ai_family* means that the caller shall accept any address family. A value of zero

for *ai_socktype* means that the caller shall accept any socket type. A value of zero for *ai_protocol* means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if it referred to a structure containing the value zero for the *ai_flags*, *ai_socktype*, and *ai_protocol* fields, and AF_UNSPEC for the *ai_family* field.

The *ai_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-inclusive OR of one or more of the values AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST, AI_NUMERICSERV, AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG.

If the AI_PASSIVE flag is specified, the returned address information shall be suitable for use in binding a socket for accepting incoming connections for the specified service. In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY_INIT for an IPv6 address. If the AI_PASSIVE flag is not specified, the returned address information shall be suitable for a call to *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a connectionless protocol). In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to the loopback address. The AI_PASSIVE flag shall be ignored if the *nodename* argument is not null.

If the AI_CANONNAME flag is specified and the *nodename* argument is not null, the function shall attempt to determine the canonical name corresponding to *nodename* (for example, if *nodename* is an alias or shorthand notation for a complete name).

Note: Since different implementations use different conceptual models, the terms “canonical name” and “alias” cannot be precisely defined for the general case. However, Domain Name System implementations are expected to interpret them as they are used in RFC 1034.

A numeric host address string is not a “name”, and thus does not have a “canonical name” form; no address to host name translation is performed. See below for handling of the case where a canonical name cannot be obtained.

If the AI_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a numeric host address string. Otherwise, an [EAI_NONAME] error is returned. This flag shall prevent any type of name resolution service (for example, the DNS) from being invoked.

If the AI_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a numeric port string. Otherwise, an [EAI_NONAME] error shall be returned. This flag shall prevent any type of name resolution service (for example, NIS+) from being invoked.

If the AI_V4MAPPED flag is specified along with an *ai_family* of AF_INET6, then *getaddrinfo()* shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (*ai_addrlen* shall be 16). The AI_V4MAPPED flag shall be ignored unless *ai_family* equals AF_INET6. If the AI_ALL flag is used with the AI_V4MAPPED flag, then *getaddrinfo()* shall return all matching IPv6 and IPv4 addresses. The AI_ALL flag without the AI_V4MAPPED flag is ignored.

If the AI_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6 address is configured on the local system.

The *ai_socktype* field to which argument *hints* points specifies the socket type for the service, as defined in *socket()*. If a specific socket type is not given (for example, a value of zero) and the service name could be interpreted as valid with multiple supported socket types, the implementation shall attempt to resolve the service name for all supported socket types and, in the absence of errors, all possible results shall be returned. A non-zero socket type value shall limit the returned information to values with the specified socket type.

If the *ai_family* field to which *hints* points has the value AF_UNSPEC, addresses shall be returned for use with any address family that can be used with the specified *nodename* and/or *servname*. Otherwise, addresses shall be returned for use only with the specified address family.

14050 If *ai_family* is not AF_UNSPEC and *ai_protocol* is not zero, then addresses are returned for use
 14051 only with the specified address family and protocol; the value of *ai_protocol* shall be interpreted
 14052 as in a call to the *socket()* function with the corresponding values of *ai_family* and *ai_protocol*.

14053 RETURN VALUE

14054 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value
 14055 indicates failure. The possible values for the failures are listed in the ERRORS section.

14056 Upon successful return of *getaddrinfo()*, the location to which *res* points shall refer to a linked list
 14057 of **addrinfo** structures, each of which shall specify a socket address and information for use in
 14058 creating a socket with which to use that socket address. The list shall include at least one
 14059 **addrinfo** structure. The *ai_next* field of each structure contains a pointer to the next structure on
 14060 the list, or a null pointer if it is the last structure on the list. Each structure on the list shall
 14061 include values for use with a call to the *socket()* function, and a socket address for use with the
 14062 *connect()* function or, if the AI_PASSIVE flag was specified, for use with the *bind()* function. The
 14063 fields *ai_family*, *ai_socktype*, and *ai_protocol* shall be usable as the arguments to the *socket()*
 14064 function to create a socket suitable for use with the returned address. The fields *ai_addr* and
 14065 *ai_addrlen* are usable as the arguments to the *connect()* or *bind()* functions with such a socket,
 14066 according to the AI_PASSIVE flag.

14067 If *nodename* is not null, and if requested by the AI_CANONNAME flag, the *ai_canonname* field of
 14068 the first returned **addrinfo** structure shall point to a null-terminated string containing the
 14069 canonical name corresponding to the input *nodename*; if the canonical name is not available, then
 14070 *ai_canonname* shall refer to the *nodename* argument or a string with the same contents. The
 14071 contents of the *ai_flags* field of the returned structures are undefined.

14072 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an
 14073 explicit argument (for example, *sin6_flowinfo*) shall be set to zero.

14074 **Note:** This makes it easier to compare socket address structures.

14075 ERRORS

14076 The *getaddrinfo()* function shall fail and return the corresponding value if:

14077 [EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

14078 [EAI_BADFLAGS]

14079 The *flags* parameter had an invalid value.

14080 [EAI_FAIL] A non-recoverable error occurred when attempting to resolve the name.

14081 [EAI_FAMILY] The address family was not recognized.

14082 [EAI_MEMORY] There was a memory allocation failure when trying to allocate storage for the
 14083 return value.

14084 [EAI_NONAME] The name does not resolve for the supplied parameters.

14085 Neither *nodename* nor *servname* were supplied. At least one of these shall be
 14086 supplied.

14087 [EAI_SERVICE] The service passed was not recognized for the specified socket type.

14088 [EAI_SOCKTYPE]

14089 The intended socket type was not recognized.

14090 [EAI_SYSTEM] A system error occurred; the error code can be found in *errno*.

14091 [EAI_OVERFLOW]

14092 An argument buffer overflowed.

14093 **EXAMPLES**

14094 None.

14095 **APPLICATION USAGE**

14096 If the caller handles only TCP and not UDP, for example, then the *ai_protocol* member of the *hints*
 14097 structure should be set to IPPROTO_TCP when *getaddrinfo()* is called.

14098 If the caller handles only IPv4 and not IPv6, then the *ai_family* member of the *hints* structure
 14099 should be set to AF_INET when *getaddrinfo()* is called.

14100 The term “canonical name” is misleading; it is taken from the Domain Name System (RFC 2181).
 14101 It should be noted that the canonical name is a result of alias processing, and not necessarily a
 14102 unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this
 14103 in the Domain Name System context.

14104 **RATIONALE**

14105 None.

14106 **FUTURE DIRECTIONS**

14107 None.

14108 **SEE ALSO**

14109 *connect()*, *gai_strerror()*, *gethostbyaddr()*, *getnameinfo()*, *getservbyname()*, *socket()*, the Base
 14110 Definitions volume of IEEE Std 1003.1-2001, <netdb.h>, <sys/socket.h>

14111 **CHANGE HISTORY**

14112 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

14113 The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the
 14114 ISO/IEC 9899:1999 standard.

14115 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the
 14116 DESCRIPTION and adding text to the APPLICATION USAGE related to the term “canonical
 14117 name”. A reference to RFC 2181 is also added to the Informative References.

14118 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for
 14119 alignment with IPv6. These include the following:

- 14120 • Adding AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG to the allowed values for the
 14121 *ai_flags* field
- 14122 • Adding a description of AI_ADDRCONFIG
- 14123 • Adding a description of the consequences of ignoring the AI_PASSIVE flag.

14124 **NAME**

14125 freopen — open a stream

14126 **SYNOPSIS**

14127 #include <stdio.h>

14128 FILE *freopen(const char *restrict *filename*, const char *restrict *mode*,
 14129 FILE *restrict *stream*);

14130 **DESCRIPTION**

14131 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14132 conflict between the requirements described here and the ISO C standard is unintentional. This
 14133 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14134 The *freopen()* function shall first attempt to flush the stream and close any file descriptor
 14135 associated with *stream*. Failure to flush or close the file descriptor successfully shall be ignored.
 14136 The error and end-of-file indicators for the stream shall be cleared.

14137 The *freopen()* function shall open the file whose pathname is the string pointed to by *filename* and
 14138 associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in
 14139 *fopen()*.

14140 The original stream shall be closed regardless of whether the subsequent open succeeds.

14141 If *filename* is a null pointer, the *freopen()* function shall attempt to change the mode of the stream
 14142 to that specified by *mode*, as if the name of the file currently associated with the stream had been
 14143 used. It is implementation-defined which changes of mode are permitted (if any), and under
 14144 what circumstances.

14145 XSI After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the
 14146 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an
 14147 initial conversion state.

14148 CX The largest value that can be represented correctly in an object of type **off_t** shall be established
 14149 as the offset maximum in the open file description.

14150 **RETURN VALUE**

14151 Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer
 14152 CX shall be returned, and *errno* shall be set to indicate the error.

14153 **ERRORS**

14154 The *freopen()* function shall fail if:

14155 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
 14156 exists and the permissions specified by *mode* are denied, or the file does not
 14157 exist and write permission is denied for the parent directory of the file to be
 14158 created.

14159 CX [EINTR] A signal was caught during *freopen()*.

14160 CX [EISDIR] The named file is a directory and *mode* requires write access.

14161 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 14162 argument.

14163 CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

14164 CX [ENAMETOOLONG] The length of the *filename* argument exceeds {PATH_MAX} or a pathname
 14165 component is longer than {NAME_MAX}.
 14166

14167 CX	[ENFILE]	The maximum allowable number of files is currently open in the system.
14168 CX 14169	[ENOENT]	A component of <i>filename</i> does not name an existing file or <i>filename</i> is an empty string.
14170 CX 14171	[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.
14172 CX	[ENOTDIR]	A component of the path prefix is not a directory.
14173 CX 14174	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
14175 CX 14176	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type off_t .
14177 CX 14178	[EROFS]	The named file resides on a read-only file system and <i>mode</i> requires write access.
14179	The <i>freopen()</i> function may fail if:	
14180 CX	[EINVAL]	The value of the <i>mode</i> argument is not valid.
14181 CX 14182	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
14183 CX 14184 14185	[ENAMETOOLONG]	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
14186 CX	[ENOMEM]	Insufficient storage space is available.
14187 CX 14188	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
14189 CX 14190	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.

14191 EXAMPLES

14192 Directing Standard Output to a File

14193 The following example logs all standard output to the **/tmp/logfile** file.

```

14194 #include <stdio.h>
14195 ...
14196 FILE *fp;
14197 ...
14198 fp = freopen ("/tmp/logfile", "a+", stdout);
14199 ...

```

14200 APPLICATION USAGE

14201 The *freopen()* function is typically used to attach the preopened *streams* associated with *stdin*,
 14202 *stdout*, and *stderr* to other files.

14203 RATIONALE

14204 None.

14205 **FUTURE DIRECTIONS**

14206 None.

14207 **SEE ALSO**

14208 *fclose()*, *fopen()*, *fdopen()*, *mbsinit()*, the Base Definitions volume of IEEE Std 1003.1-2001,
14209 **<stdio.h>**

14210 **CHANGE HISTORY**

14211 First released in Issue 1. Derived from Issue 1 of the SVID.

14212 **Issue 5**

14213 The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the
14214 conversion state of the stream is set to an initial conversion state by a successful call to the
14215 *freopen()* function.

14216 Large File Summit extensions are added.

14217 **Issue 6**

14218 Extensions beyond the ISO C standard are marked.

14219 The following new requirements on POSIX implementations derive from alignment with the
14220 Single UNIX Specification:

- 14221 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
14222 description. This change is to support large files.
- 14223 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
14224 large files.
- 14225 • The [ELOOP] mandatory error condition is added.
- 14226 • A second [ENAMETOOLONG] is added as an optional error condition.
- 14227 • The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

14228 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 14229 • The *freopen()* prototype is updated.
- 14230 • The DESCRIPTION is updated.

14231 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
14232 [ELOOP] error condition is added.

14233 The DESCRIPTION is updated regarding failure to close, changing the “file” to “file descriptor”.

14234 **NAME**

14235 frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

14236 **SYNOPSIS**

14237 #include <math.h>

14238 double frexp(double num, int *exp);

14239 float frexpf(float num, int *exp);

14240 long double frexpl(long double num, int *exp);

14241 **DESCRIPTION**

14242 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14243 conflict between the requirements described here and the ISO C standard is unintentional. This
 14244 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14245 These functions shall break a floating-point number *num* into a normalized fraction and an
 14246 integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.

14247 **RETURN VALUE**

14248 For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the
 14249 interval $[\frac{1}{2}, 1)$ or 0, and *num* equals *x* times 2 raised to the power **exp*.

14250 MX If *num* is NaN, a NaN shall be returned, and the value of **exp* is unspecified.

14251 If *num* is ± 0 , ± 0 shall be returned, and the value of **exp* shall be 0.

14252 If *num* is $\pm \text{Inf}$, *num* shall be returned, and the value of **exp* is unspecified.

14253 **ERRORS**

14254 No errors are defined.

14255 **EXAMPLES**

14256 None.

14257 **APPLICATION USAGE**

14258 None.

14259 **RATIONALE**

14260 None.

14261 **FUTURE DIRECTIONS**

14262 None.

14263 **SEE ALSO**14264 *isnan()*, *ldexp()*, *modf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>14265 **CHANGE HISTORY**

14266 First released in Issue 1. Derived from Issue 1 of the SVID.

14267 **Issue 5**

14268 The DESCRIPTION is updated to indicate how an application should check for an error. This
 14269 text was previously published in the APPLICATION USAGE section.

14270 **Issue 6**

14271 The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999
 14272 standard.

14273	The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.
14274	
14275	IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.
14276	

14277 **NAME**

14278 fscanf, scanf, sscanf — convert formatted input

14279 **SYNOPSIS**

14280 #include <stdio.h>

14281 int fscanf(FILE *restrict *stream*, const char *restrict *format*, ...);14282 int scanf(const char *restrict *format*, ...);14283 int sscanf(const char *restrict *s*, const char *restrict *format*, ...);14284 **DESCRIPTION**

14285 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14286 conflict between the requirements described here and the ISO C standard is unintentional. This
 14287 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14288 The *fscanf()* function shall read from the named input *stream*. The *scanf()* function shall read
 14289 from the standard input stream *stdin*. The *sscanf()* function shall read from the string *s*. Each
 14290 function reads bytes, interprets them according to a format, and stores the results in its
 14291 arguments. Each expects, as arguments, a control string *format* described below, and a set of
 14292 *pointer* arguments indicating where the converted input should be stored. The result is
 14293 undefined if there are insufficient arguments for the format. If the format is exhausted while
 14294 arguments remain, the excess arguments shall be evaluated but otherwise ignored.

14295 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 14296 to the next unused argument. In this case, the conversion specifier character % (see below) is
 14297 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}].
 14298 This feature provides for the definition of format strings that select arguments in an order
 14299 appropriate to specific languages. In format strings containing the "%n\$" form of conversion
 14300 specifications, it is unspecified whether numbered arguments in the argument list can be
 14301 referenced from the format string more than once.

14302 The *format* can contain either form of a conversion specification—that is, % or "%n\$"—but the
 14303 two forms cannot be mixed within a single *format* string. The only exception to this is that %% or
 14304 %* can be mixed with the "%n\$" form. When numbered argument specifications are used,
 14305 specifying the *N*th argument requires that all the leading arguments, from the first to the
 14306 (*N*−1)th, are pointers.

14307 CX The *fscanf()* function in all its forms shall allow detection of a language-dependent radix
 14308 character in the input string. The radix character is defined in the program's locale (category
 14309 LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the
 14310 radix character shall default to a period ('.').

14311 The format is a character string, beginning and ending in its initial shift state, if any, composed
 14312 of zero or more directives. Each directive is composed of one of the following: one or more
 14313 white-space characters (<space>s, <tab>s, <newline>s, <vertical-tab>s, or <form-feed>s); an
 14314 ordinary character (neither '%' nor a white-space character); or a conversion specification. Each
 14315 XSI conversion specification is introduced by the character '%' or the character sequence "%n\$",
 14316 after which the following appear in sequence:

- 14317 • An optional assignment-suppressing character '*'.
- 14318 • An optional non-zero decimal integer that specifies the maximum field width.
- 14319 • An option length modifier that specifies the size of the receiving object.
- 14320 • A *conversion specifier* character that specifies the type of conversion to be applied. The valid
 14321 conversion specifiers are described below.

14322 The *fscanf()* functions shall execute each directive of the format in turn. If a directive fails, as
 14323 detailed below, the function shall return. Failures are described as input failures (due to the
 14324 unavailability of input bytes) or matching failures (due to inappropriate input).

14325 A directive composed of one or more white-space characters shall be executed by reading input
 14326 until no more valid input can be read, or up to the first byte which is not a white-space character,
 14327 which remains unread.

14328 A directive that is an ordinary character shall be executed as follows: the next byte shall be read
 14329 from the input and compared with the byte that comprises the directive; if the comparison
 14330 shows that they are not equivalent, the directive shall fail, and the differing and subsequent
 14331 bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a
 14332 character from being read, the directive shall fail.

14333 A directive that is a conversion specification defines a set of matching input sequences, as
 14334 described below for each conversion character. A conversion specification shall be executed in
 14335 the following steps.

14336 Input white-space characters (as specified by *isspace()*) shall be skipped, unless the conversion
 14337 specification includes a *l*, *c*, *C*, or *n* conversion specifier.

14338 An item shall be read from the input, unless the conversion specification includes an *n*
 14339 conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to
 14340 any specified maximum field width, which may be measured in characters or bytes dependent
 14341 on the conversion specifier) which is an initial subsequence of a matching sequence. The first
 14342 byte, if any, after the input item shall remain unread. If the length of the input item is 0, the
 14343 execution of the conversion specification shall fail; this condition is a matching failure, unless
 14344 end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is
 14345 an input failure.

14346 Except in the case of a *%* conversion specifier, the input item (or, in the case of a *%n* conversion
 14347 specification, the count of input bytes) shall be converted to a type appropriate to the conversion
 14348 character. If the input item is not a matching sequence, the execution of the conversion
 14349 specification fails; this condition is a matching failure. Unless assignment suppression was
 14350 indicated by a *'*'*, the result of the conversion shall be placed in the object pointed to by the
 14351 first argument following the *format* argument that has not already received a conversion result if
 14352 xsl the conversion specification is introduced by *%*, or in the *n*th argument if introduced by the
 14353 character sequence *"%n\$"*. If this object does not have an appropriate type, or if the result of the
 14354 conversion cannot be represented in the space provided, the behavior is undefined.

14355 The length modifiers and their meanings are:

14356 hh Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an
 14357 argument with type pointer to **signed char** or **unsigned char**.

14358 h Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an
 14359 argument with type pointer to **short** or **unsigned short**.

14360 l (ell) Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an
 14361 argument with type pointer to **long** or **unsigned long**; that a following *a*, *A*, *e*, *E*, *f*, *F*, *g*,
 14362 or *G* conversion specifier applies to an argument with type pointer to **double**; or that a
 14363 following *c*, *s*, or *l* conversion specifier applies to an argument with type pointer to
 14364 **wchar_t**.

14365 ll (ell-ell) Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an
 14366 argument with type pointer to **long long** or **unsigned long long**.
 14367

14368	j	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to intmax_t or uintmax_t .
14369		
14370	z	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to size_t or the corresponding signed integer type.
14371		
14372	t	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to ptrdiff_t or the corresponding unsigned type.
14373		
14374	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to long double .
14375		
14376		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
14377		
14378		The following conversion specifiers are valid:
14379	d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
14380		
14381		
14382		
14383	i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
14384		
14385		
14386		
14387	o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
14388		
14389		
14390		
14391	u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
14392		
14393		
14394		
14395	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
14396		
14397		
14398		
14399	a, e, f, g	
14400		Matches an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of <i>strtod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to float .
14401		
14402		
14403		
14404		If the <i>fprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the <i>fscanf()</i> family of functions shall recognize them as input.
14405		
14406		
14407	s	Matches a sequence of bytes that are not white-space characters. The application shall ensure that the corresponding argument is a pointer to the initial byte of an array of char , signed char , or unsigned char large enough to accept the sequence and a terminating null character code, which shall be added automatically.
14408		
14409		
14410		
14411		If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character shall be converted to a wide character as if by a call to
14412		

14413		the <i>mbrtowc()</i> function, with the conversion state described by an mbstate_t object
14414		initialized to zero before the first character is converted. The application shall ensure
14415		that the corresponding argument is a pointer to an array of wchar_t large enough to
14416		accept the sequence and the terminating null wide character, which shall be added
14417		automatically.
14418	[	Matches a non-empty sequence of bytes from a set of expected bytes (the <i>scanset</i>). The
14419		normal skip over white-space characters shall be suppressed in this case. The
14420		application shall ensure that the corresponding argument is a pointer to the initial byte
14421		of an array of char , signed char , or unsigned char large enough to accept the sequence
14422		and a terminating null byte, which shall be added automatically.
14423		If an l (ell) qualifier is present, the input is a sequence of characters that begins in the
14424		initial shift state. Each character in the sequence shall be converted to a wide character
14425		as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an
14426		mbstate_t object initialized to zero before the first character is converted. The
14427		application shall ensure that the corresponding argument is a pointer to an array of
14428		wchar_t large enough to accept the sequence and the terminating null wide character,
14429		which shall be added automatically.
14430		The conversion specification includes all subsequent bytes in the <i>format</i> string up to
14431		and including the matching right square bracket (']'). The bytes between the square
14432		brackets (the <i>scanlist</i>) comprise the scanset, unless the byte after the left square bracket
14433		is a circumflex ('^'), in which case the scanset contains all bytes that do not appear in
14434		the scanlist between the circumflex and the right square bracket. If the conversion
14435		specification begins with "[^]" or "[^]", the right square bracket is included in the
14436		scanlist and the next right square bracket is the matching right square bracket that ends
14437		the conversion specification; otherwise, the first right square bracket is the one that
14438		ends the conversion specification. If a '-' is in the scanlist and is not the first character,
14439		nor the second where the first character is a '^', nor the last character, the behavior is
14440		implementation-defined.
14441	c	Matches a sequence of bytes of the number specified by the field width (1 if no field
14442		width is present in the conversion specification). The application shall ensure that the
14443		corresponding argument is a pointer to the initial byte of an array of char , signed char ,
14444		or unsigned char large enough to accept the sequence. No null byte is added. The
14445		normal skip over white-space characters shall be suppressed in this case.
14446		If an l (ell) qualifier is present, the input shall be a sequence of characters that begins in
14447		the initial shift state. Each character in the sequence is converted to a wide character as
14448		if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an
14449		mbstate_t object initialized to zero before the first character is converted. The
14450		application shall ensure that the corresponding argument is a pointer to an array of
14451		wchar_t large enough to accept the resulting sequence of wide characters. No null wide
14452		character is added.
14453	p	Matches an implementation-defined set of sequences, which shall be the same as the set
14454		of sequences that is produced by the %p conversion specification of the corresponding
14455		<i>fprintf()</i> functions. The application shall ensure that the corresponding argument is a
14456		pointer to a pointer to void . The interpretation of the input item is implementation-
14457		defined. If the input item is a value converted earlier during the same program
14458		execution, the pointer that results shall compare equal to that value; otherwise, the
14459		behavior of the %p conversion specification is undefined.
14460	n	No input is consumed. The application shall ensure that the corresponding argument is
14461		a pointer to the integer into which shall be written the number of bytes read from the

14462 input so far by this call to the *fscanf()* functions. Execution of a %n conversion
 14463 specification shall not increment the assignment count returned at the completion of
 14464 execution of the function. No argument shall be converted, but one shall be consumed.
 14465 If the conversion specification includes an assignment-suppressing character or a field
 14466 width, the behavior is undefined.

14467 XSI C Equivalent to lc.

14468 XSI S Equivalent to ls.

14469 % Matches a single '%' character; no conversion or assignment occurs. The complete
 14470 conversion specification shall be %%.

14471 If a conversion specification is invalid, the behavior is undefined.

14472 The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to a, e, f, g, and
 14473 x, respectively.

14474 If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs
 14475 before any bytes matching the current conversion specification (except for %n) have been read
 14476 (other than leading white-space characters, where permitted), execution of the current
 14477 conversion specification shall terminate with an input failure. Otherwise, unless execution of the
 14478 current conversion specification is terminated with a matching failure, execution of the
 14479 following conversion specification (if any) shall be terminated with an input failure.

14480 Reaching the end of the string in *sscanf()* shall be equivalent to encountering end-of-file for
 14481 *fscanf()*.

14482 If conversion terminates on a conflicting input, the offending input is left unread in the input.
 14483 Any trailing white space (including <newline>s) shall be left unread unless matched by a
 14484 conversion specification. The success of literal matches and suppressed assignments is only
 14485 directly determinable via the %n conversion specification.

14486 CX The *fscanf()* and *scanf()* functions may mark the *st_atime* field of the file associated with *stream*
 14487 for update. The *st_atime* field shall be marked for update by the first successful execution of
 14488 *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()*, *fscanf()*, or *scanf()* using *stream* that returns data
 14489 not supplied by a prior call to *ungetc()*.

14490 RETURN VALUE

14491 Upon successful completion, these functions shall return the number of successfully matched
 14492 and assigned input items; this number can be zero in the event of an early matching failure. If
 14493 the input ends before the first matching failure or conversion, EOF shall be returned. If a read
 14494 CX error occurs, the error indicator for the stream is set, EOF shall be returned, and *errno* shall be set
 14495 to indicate the error.

14496 ERRORS

14497 For the conditions under which the *fscanf()* functions fail and may fail, refer to *fgetc()* or
 14498 *fgetwc()*.

14499 In addition, *fscanf()* may fail if:

14500 XSI [EILSEQ] Input byte sequence does not form a valid character.

14501 XSI [EINVAL] There are insufficient arguments.

14502 **EXAMPLES**

14503 The call:

```
14504     int i, n; float x; char name[50];
14505     n = scanf("%d%f%s", &i, &x, name);
```

14506 with the input line:

14507 25 54.32E-1 Hamster

14508 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string
 14509 "Hamster".

14510 The call:

```
14511     int i; float x; char name[50];
14512     (void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

14513 with input:

14514 56789 0123 56a72

14515 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to
 14516 *getchar()* shall return the character 'a'.

14517 **Reading Data into an Array**

14518 The following call uses *fscanf()* to read three floating-point numbers from standard input into
 14519 the *input* array.

```
14520     float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

14521 **APPLICATION USAGE**

14522 If the application calling *fscanf()* has any objects of type **wint_t** or **wchar_t**, it must also include
 14523 the **<wchar.h>** header to have these objects defined.

14524 **RATIONALE**

14525 This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few “obvious”
 14526 things were not included. Specifically, the set of characters allowed in a scanset is limited to
 14527 single-byte characters. In other similar places, multi-byte characters have been permitted, but
 14528 for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications
 14529 needing this could use the corresponding wide-character functions to achieve the desired
 14530 results.

14531 **FUTURE DIRECTIONS**

14532 None.

14533 **SEE ALSO**

14534 *getc()*, *printf()*, *setlocale()*, *strtod()*, *strtol()*, *strtoul()*, *wcrtomb()*, the Base Definitions volume of
 14535 IEEE Std 1003.1-2001, Chapter 7, Locale, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

14536 **CHANGE HISTORY**

14537 First released in Issue 1. Derived from Issue 1 of the SVID.

14538 **Issue 5**

14539 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the *l* (ell) qualifier is
 14540 now defined for the *c*, *s*, and *[* conversion specifiers.

14541 The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the
 14542 *fprintf()* family of functions, then they are recognized by the *fscanf()* family.

14543 **Issue 6**

14544 The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several
14545 occurrences of “characters” in the text which have been replaced with the term “bytes”.

14546 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14547 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 14548 • The prototypes for *fscanf()*, *scanf()*, and *sscanf()* are updated.
- 14549 • The DESCRIPTION is updated.
- 14550 • The hh, ll, j, t, and z length modifiers are added.
- 14551 • The a, A, and F conversion characters are added.

14552 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
14553 specification” consistently.

14554 **NAME**

14555 fseek, fseeko — reposition a file-position indicator in a stream

14556 **SYNOPSIS**

14557 #include <stdio.h>

14558 int fseek(FILE *stream, long offset, int whence);

14559 CX int fseeko(FILE *stream, off_t offset, int whence);

14560

14561 **DESCRIPTION**

14562 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14563 conflict between the requirements described here and the ISO C standard is unintentional. This
 14564 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14565 The *fseek()* function shall set the file-position indicator for the stream pointed to by *stream*. If a
 14566 read or write error occurs, the error indicator for the stream shall be set and *fseek()* fails.

14567 The new position, measured in bytes from the beginning of the file, shall be obtained by adding
 14568 *offset* to the position specified by *whence*. The specified point is the beginning of the file for
 14569 SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for
 14570 SEEK_END.

14571 If the stream is to be used with wide-character input/output functions, the application shall
 14572 ensure that *offset* is either 0 or a value returned by an earlier call to *ftell()* on the same stream and
 14573 *whence* is SEEK_SET.

14574 A successful call to *fseek()* shall clear the end-of-file indicator for the stream and undo any effects
 14575 of *ungetc()* and *ungetwc()* on the same stream. After an *fseek()* call, the next operation on an
 14576 update stream may be either input or output.

14577 CX If the most recent operation, other than *ftell()*, on a given stream is *fflush()*, the file offset in the
 14578 underlying open file description shall be adjusted to reflect the location specified by *fseek()*.

14579 The *fseek()* function shall allow the file-position indicator to be set beyond the end of existing
 14580 data in the file. If data is later written at this point, subsequent reads of data in the gap shall
 14581 return bytes with the value 0 until data is actually written into the gap.

14582 The behavior of *fseek()* on devices which are incapable of seeking is implementation-defined.
 14583 The value of the file offset associated with such a device is undefined.

14584 If the stream is writable and buffered data had not been written to the underlying file, *fseek()*
 14585 shall cause the unwritten data to be written to the file and shall mark the *st_ctime* and *st_mtime*
 14586 fields of the file for update.

14587 In a locale with state-dependent encoding, whether *fseek()* restores the stream's shift state is
 14588 implementation-defined.

14589 The *fseeko()* function shall be equivalent to the *fseek()* function except that the *offset* argument is
 14590 of type **off_t**.

14591 **RETURN VALUE**14592 CX The *fseek()* and *fseeko()* functions shall return 0 if they succeed.14593 CX Otherwise, they shall return -1 and set *errno* to indicate the error.14594 **ERRORS**

14595 CX The *fseek()* and *fseeko()* functions shall fail if, either the *stream* is unbuffered or the *stream*'s
 14596 buffer needed to be flushed, and the call to *fseek()* or *fseeko()* causes an underlying *lseek()* or
 14597 *write()* to be invoked, and:

14598 CX	[EAGAIN]	The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in the write operation.
14599		
14600 CX	[EBADF]	The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.
14601		
14602 CX	[EFBIG]	An attempt was made to write a file that exceeds the maximum file size.
14603 XSI	[EFBIG]	An attempt was made to write a file that exceeds the process' file size limit.
14604 CX	[EFBIG]	The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.
14605		
14606 CX	[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
14607		
14608 CX	[EINVAL]	The <i>whence</i> argument is invalid. The resulting file-position indicator would be set to a negative value.
14609		
14610 CX	[EIO]	A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a <i>write()</i> to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
14611		
14612		
14613		
14614		
14615 CX	[ENOSPC]	There was no free space remaining on the device containing the file.
14616 CX	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
14617		
14618 CX	[EOVERFLOW]	For <i>fseek()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type long .
14619		
14620 CX	[EOVERFLOW]	For <i>fseeko()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type off_t .
14621		
14622 CX	[EPIPE]	An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.
14623		
14624 CX	[ESPIPE]	The file descriptor underlying <i>stream</i> is associated with a pipe or FIFO.

14625 **EXAMPLES**

14626 None.

14627 **APPLICATION USAGE**

14628 None.

14629 **RATIONALE**

14630 None.

14631 **FUTURE DIRECTIONS**

14632 None.

14633 **SEE ALSO**

14634 *fopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *lseek()*, *rewind()*, *ulimit()*, *ungetc()*, *write()*, the Base
 14635 Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>

14636 **CHANGE HISTORY**

14637 First released in Issue 1. Derived from Issue 1 of the SVID.

14638 **Issue 5**

14639 Normative text previously in the APPLICATION USAGE section is moved to the
14640 DESCRIPTION.

14641 Large File Summit extensions are added.

14642 **Issue 6**

14643 Extensions beyond the ISO C standard are marked.

14644 The following new requirements on POSIX implementations derive from alignment with the
14645 Single UNIX Specification:

- 14646 • The *fseeko()* function is added.
- 14647 • The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

14648 The following change is incorporated for alignment with the FIPS requirements:

- 14649 • The [EINTR] error is no longer an indication that the implementation does not report partial
14650 transfers.

14651 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14652 The DESCRIPTION is updated to explicitly state that *fseek()* sets the file-position indicator, and
14653 then on error the error indicator is set and *fseek()* fails. This is for alignment with the
14654 ISO / IEC 9899:1999 standard.

14655 **NAME**

14656 fsetpos — set current file position

14657 **SYNOPSIS**

14658 #include <stdio.h>

14659 int fsetpos(FILE *stream, const fpos_t *pos);

14660 **DESCRIPTION**

14661 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14662 conflict between the requirements described here and the ISO C standard is unintentional. This
 14663 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14664 The *fsetpos()* function shall set the file position and state indicators for the stream pointed to by
 14665 *stream* according to the value of the object pointed to by *pos*, which the application shall ensure
 14666 is a value obtained from an earlier call to *fgetpos()* on the same stream. If a read or write error
 14667 occurs, the error indicator for the stream shall be set and *fsetpos()* fails.

14668 A successful call to the *fsetpos()* function shall clear the end-of-file indicator for the stream and
 14669 undo any effects of *ungetc()* on the same stream. After an *fsetpos()* call, the next operation on an
 14670 update stream may be either input or output.

14671 CX The behavior of *fsetpos()* on devices which are incapable of seeking is implementation-defined.
 14672 The value of the file offset associated with such a device is undefined.

14673 **RETURN VALUE**

14674 The *fsetpos()* function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and
 14675 set *errno* to indicate the error.

14676 **ERRORS**

14677 CX The *fsetpos()* function shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to
 14678 be flushed, and the call to *fsetpos()* causes an underlying *lseek()* or *write()* to be invoked, and:

14679 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the process would be
 14680 delayed in the write operation.

14681 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the
 14682 stream's buffer needed to be flushed and the file is not open.

14683 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

14684 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

14685 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 14686 offset maximum associated with the corresponding stream.

14687 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 14688 was transferred.

14689 CX [EIO] A physical I/O error has occurred, or the process is a member of a
 14690 background process group attempting to perform a *write()* to its controlling
 14691 terminal, TOSTOP is set, the process is neither ignoring nor blocking
 14692 SIGTTOU, and the process group of the process is orphaned. This error may
 14693 also be returned under implementation-defined conditions.

14694 CX [ENOSPC] There was no free space remaining on the device containing the file.

14695 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 14696 capabilities of the device.

14697 CX [EPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

14698 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading

14699 by any process; a SIGPIPE signal shall also be sent to the thread.

14700 EXAMPLES

14701 None.

14702 APPLICATION USAGE

14703 None.

14704 RATIONALE

14705 None.

14706 FUTURE DIRECTIONS

14707 None.

14708 SEE ALSO

14709 *fopen()*, *ftell()*, *lseek()*, *rewind()*, *ungetc()*, *write()*, the Base Definitions volume of

14710 IEEE Std 1003.1-2001, <stdio.h>

14711 CHANGE HISTORY

14712 First released in Issue 4. Derived from the ISO C standard.

14713 Issue 6

14714 Extensions beyond the ISO C standard are marked.

14715 An additional [ESPIPE] error condition is added for sockets.

14716 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14717 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or

14718 write error. This is for alignment with the ISO/IEC 9899:1999 standard.

14719 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous |

14720 [EINVAL] error case from the ERRORS section. |

14721 **NAME**

14722 fstat — get file status

14723 **SYNOPSIS**

14724 #include <sys/stat.h>

14725 int fstat(int *fildes*, struct stat **buf*);14726 **DESCRIPTION**

14727 The *fstat()* function shall obtain information about an open file associated with the file
 14728 descriptor *fildes*, and shall write it to the area pointed to by *buf*.

14729 SHM If *fildes* references a shared memory object, the implementation shall update in the **stat** structure
 14730 pointed to by the *buf* argument only the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
 14731 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
 14732 valid. The implementation may update other fields and flags.

14733 TYM If *fildes* references a typed memory object, the implementation shall update in the **stat** structure
 14734 pointed to by the *buf* argument only the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
 14735 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
 14736 valid. The implementation may update other fields and flags.

14737 The *buf* argument is a pointer to a **stat** structure, as defined in <sys/stat.h>, into which
 14738 information is placed concerning the file.

14739 The structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*, *st_ctime*, and *st_mtime*
 14740 shall have meaningful values for all other file types defined in this volume of
 14741 IEEE Std 1003.1-2001. The value of the member *st_nlink* shall be set to the number of links to the
 14742 file.

14743 An implementation that provides additional or alternative file access control mechanisms may,
 14744 under implementation-defined conditions, cause *fstat()* to fail.

14745 The *fstat()* function shall update any time-related fields as described in the Base Definitions
 14746 volume of IEEE Std 1003.1-2001, Section 4.7, File Times Update, before writing into the **stat**
 14747 structure.

14748 **RETURN VALUE**

14749 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 14750 indicate the error.

14751 **ERRORS**14752 The *fstat()* function shall fail if:

14753 [EBADF] The *fildes* argument is not a valid file descriptor.

14754 [EIO] An I/O error occurred while reading from the file system.

14755 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
 14756 serial number cannot be represented correctly in the structure pointed to by
 14757 *buf*.

14758 The *fstat()* function may fail if:

14759 [EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf*
 14760 argument.

14761 **EXAMPLES**14762 **Obtaining File Status Information**

14763 The following example shows how to obtain file status information for a file named
 14764 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The
 14765 **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file
 14766 descriptor *fildes*.

```
14767 #include <sys/types.h>
14768 #include <sys/stat.h>
14769 #include <fcntl.h>

14770 struct stat buffer;
14771 int      status;
14772 ...
14773 fildes = open("/home/cnd/mod1", O_RDWR);
14774 status = fstat(fildes, &buffer);
```

14775 **APPLICATION USAGE**

14776 None.

14777 **RATIONALE**

14778 None.

14779 **FUTURE DIRECTIONS**

14780 None.

14781 **SEE ALSO**

14782 *lstat()*, *stat()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**, **<sys/types.h>**

14783 **CHANGE HISTORY**

14784 First released in Issue 1. Derived from Issue 1 of the SVID.

14785 **Issue 5**

14786 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

14787 Large File Summit extensions are added.

14788 **Issue 6**

14789 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

14790 The following new requirements on POSIX implementations derive from alignment with the
 14791 Single UNIX Specification:

- 14792 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
 14793 required for conforming implementations of previous POSIX specifications, it was not
 14794 required for UNIX applications.
- 14795 • The [EIO] mandatory error condition is added.
- 14796 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
 14797 files.
- 14798 • The [EOVERFLOW] optional error condition is added.

14799 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 14800 shared memory object semantics apply to typed memory objects.

14801 **NAME**

14802 fstatvfs, statvfs — get file system information

14803 **SYNOPSIS**

14804 xSI #include <sys/statvfs.h>

14805 int fstatvfs(int *fildes*, struct statvfs **buf*);14806 int statvfs(const char *restrict *path*, struct statvfs *restrict *buf*);

14807

14808 **DESCRIPTION**14809 The *fstatvfs()* function shall obtain information about the file system containing the file
14810 referenced by *fildes*.14811 The *statvfs()* function shall obtain information about the file system containing the file named by
14812 *path*.14813 For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read,
14814 write, or execute permission of the named file is not required.14815 The following flags can be returned in the *f_flag* member:

14816 ST_RDONLY Read-only file system.

14817 ST_NOSUID Setuid/setgid bits ignored by *exec*.14818 It is unspecified whether all members of the **statvfs** structure have meaningful values on all file
14819 systems.14820 **RETURN VALUE**14821 Upon successful completion, *statvfs()* shall return 0. Otherwise, it shall return -1 and set *errno* to
14822 indicate the error.14823 **ERRORS**14824 The *fstatvfs()* and *statvfs()* functions shall fail if:

14825 [EIO] An I/O error occurred while reading the file system.

14826 [EINTR] A signal was caught during execution of the function.

14827 [EOVERFLOW] One of the values to be returned cannot be represented correctly in the
14828 structure pointed to by *buf*.14829 The *fstatvfs()* function shall fail if:14830 [EBADF] The *fildes* argument is not an open file descriptor.14831 The *statvfs()* function shall fail if:

14832 [EACCES] Search permission is denied on a component of the path prefix.

14833 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
14834 argument.

14835 [ENAMETOOLONG]

14836 The length of a pathname exceeds {PATH_MAX} or a pathname component is
14837 longer than {NAME_MAX}.14838 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.14839 [ENOTDIR] A component of the path prefix of *path* is not a directory.

14840 The *statvfs()* function may fail if:

14841 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
14842 resolution of the *path* argument.

14843 [ENAMETOOLONG]

14844 Pathname resolution of a symbolic link produced an intermediate result
14845 whose length exceeds {PATH_MAX}.

14846 EXAMPLES

14847 Obtaining File System Information Using *fstatvfs()*

14848 The following example shows how to obtain file system information for the file system upon
14849 which the file named **/home/cnd/mod1** resides, using the *fstatvfs()* function. The
14850 **/home/cnd/mod1** file is opened with read/write privileges and the open file descriptor is passed
14851 to the *fstatvfs()* function.

```
14852 #include <statvfs.h>
```

```
14853 #include <fcntl.h>
```

```
14854 struct statvfs buffer;
```

```
14855 int status;
```

```
14856 ...
```

```
14857 fildes = open("/home/cnd/mod1", O_RDWR);
```

```
14858 status = fstatvfs(fildes, &buffer);
```

14859 Obtaining File System Information Using *statvfs()*

14860 The following example shows how to obtain file system information for the file system upon
14861 which the file named **/home/cnd/mod1** resides, using the *statvfs()* function.

```
14862 #include <statvfs.h>
```

```
14863 struct statvfs buffer;
```

```
14864 int status;
```

```
14865 ...
```

```
14866 status = statvfs("/home/cnd/mod1", &buffer);
```

14867 APPLICATION USAGE

14868 None.

14869 RATIONALE

14870 None.

14871 FUTURE DIRECTIONS

14872 None.

14873 SEE ALSO

14874 *chmod()*, *chown()*, *creat()*, *dup()*, *exec*, *fcntl()*, *link()*, *mknod()*, *open()*, *pipe()*, *read()*, *time()*,
14875 *unlink()*, *utime()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/statvfs.h**>

14876 CHANGE HISTORY

14877 First released in Issue 4, Version 2.

14878 Issue 5

14879 Moved from X/OPEN UNIX extension to BASE.

14880 Large File Summit extensions are added.

14881 **Issue 6**

- 14882 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 14883 The **restrict** keyword is added to the *statvfs()* prototype for alignment with the
- 14884 ISO/IEC 9899:1999 standard.
- 14885 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
- 14886 [ELOOP] error condition is added.

14887 **NAME**

14888 fsync — synchronize changes to a file

14889 **SYNOPSIS**14890 FSC `#include <unistd.h>`14891 `int fsync(int fildes);`

14892

14893 **DESCRIPTION**

14894 The *fsync()* function shall request that all data for the open file descriptor named by *fil*des is to be
 14895 transferred to the storage device associated with the file described by *fil*des in an
 14896 implementation-defined manner. The *fsync()* function shall not return until the system has
 14897 completed that action or until an error is detected.

14898 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued
 14899 I/O operations associated with the file indicated by file descriptor *fil*des to the synchronized I/O
 14900 completion state. All I/O operations shall be completed as defined for synchronized I/O file
 14901 integrity completion.

14902 **RETURN VALUE**

14903 Upon successful completion, *fsync()* shall return 0. Otherwise, `-1` shall be returned and *errno* set
 14904 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed
 14905 to have been completed.

14906 **ERRORS**14907 The *fsync()* function shall fail if:14908 [EBADF] The *fil*des argument is not a valid descriptor.14909 [EINTR] The *fsync()* function was interrupted by a signal.14910 [EINVAL] The *fil*des argument does not refer to a file on which this operation is possible.

14911 [EIO] An I/O error occurred while reading from or writing to the file system.

14912 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions
 14913 defined for *read()* and *write()*.

14914 **EXAMPLES**

14915 None.

14916 **APPLICATION USAGE**

14917 The *fsync()* function should be used by programs which require modifications to a file to be
 14918 completed before continuing; for example, a program which contains a simple transaction
 14919 facility might use it to ensure that all modifications to a file or files caused by a transaction are
 14920 recorded.

14921 **RATIONALE**

14922 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to
 14923 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is
 14924 recorded on the disk. Since the concepts of “buffer cache”, “system crash”, “physical write”, and
 14925 “non-volatile storage” are not defined here, the wording has to be more abstract.

14926 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance
 14927 document to tell the user what can be expected from the system. It is explicitly intended that a
 14928 null implementation is permitted. This could be valid in the case where the system cannot assure
 14929 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the
 14930 functionality is not required. In the middle ground between these extremes, *fsync()* might or
 14931 might not actually cause data to be written where it is safe from a power failure. The

14932 conformance document should identify at least that one configuration exists (and how to obtain
14933 that configuration) where this can be assured for at least some files that the user can select to use
14934 for critical data. It is not intended that an exhaustive list is required, but rather sufficient
14935 information is provided so that if critical data needs to be saved, the user can determine how the
14936 system is to be configured to allow the data to be written to non-volatile storage.

14937 It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite.
14938 That does not make the function any less valuable, just more difficult to test. A formal
14939 conformance test should probably force a system crash (power shutdown) during the test for
14940 this condition, but it needs to be done in such a way that automated testing does not require this
14941 to be done except when a formal record of the results is being made. It would also not be
14942 unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation
14943 issue.

14944 **FUTURE DIRECTIONS**

14945 None.

14946 **SEE ALSO**

14947 *sync()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

14948 **CHANGE HISTORY**

14949 First released in Issue 3.

14950 **Issue 5**

14951 Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and
14952 RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate
14953 that *fsync()* can return the error conditions defined for *read()* and *write()*.

14954 **Issue 6**

14955 This function is marked as part of the File Synchronization option.

14956 The following new requirements on POSIX implementations derive from alignment with the
14957 Single UNIX Specification:

- 14958 • The [EINVAL] and [EIO] mandatory error conditions are added.

14959 **NAME**

14960 ftell, ftello — return a file offset in a stream

14961 **SYNOPSIS**

14962 #include <stdio.h>

14963 long ftell(FILE *stream);

14964 CX off_t ftello(FILE *stream);

14965

14966 **DESCRIPTION**

14967 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14968 conflict between the requirements described here and the ISO C standard is unintentional. This
 14969 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14970 The *ftell()* function shall obtain the current value of the file-position indicator for the stream
 14971 pointed to by *stream*.

14972 CX The *ftello()* function shall be equivalent to *ftell()*, except that the return value is of type **off_t**.

14973 **RETURN VALUE**

14974 CX Upon successful completion, *ftell()* and *ftello()* shall return the current value of the file-position
 14975 indicator for the stream measured in bytes from the beginning of the file.

14976 CX Otherwise, *ftell()* and *ftello()* shall return -1 , cast to **long** and **off_t** respectively, and set *errno* to
 14977 indicate the error.

14978 **ERRORS**

14979 CX The *ftell()* and *ftello()* functions shall fail if:

14980 CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.

14981 CX [EOVERFLOW] For *ftell()*, the current file offset cannot be represented correctly in an object of
 14982 type **long**.

14983 CX [EOVERFLOW] For *ftello()*, the current file offset cannot be represented correctly in an object
 14984 of type **off_t**.

14985 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

14986 The *ftell()* function may fail if:

14987 CX [ESPIPE] The file descriptor underlying *stream* is associated with a socket.

14988 **EXAMPLES**

14989 None.

14990 **APPLICATION USAGE**

14991 None.

14992 **RATIONALE**

14993 None.

14994 **FUTURE DIRECTIONS**

14995 None.

14996 **SEE ALSO**

14997 *fgetpos()*, *fopen()*, *fseek()*, *lseek()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

14998 **CHANGE HISTORY**

14999 First released in Issue 1. Derived from Issue 1 of the SVID.

15000 **Issue 5**

15001 Large File Summit extensions are added.

15002 **Issue 6**

15003 Extensions beyond the ISO C standard are marked.

15004 The following new requirements on POSIX implementations derive from alignment with the
15005 Single UNIX Specification:

- 15006 • The *ftello()* function is added.
- 15007 • The [EOVERFLOW] error conditions are added.

15008 An additional [ESPIPE] error condition is added for sockets.

15009 NAME

15010 ftime — get date and time (**LEGACY**)

15011 SYNOPSIS

15012 xSI `#include <sys/timeb.h>`

15013 `int ftime(struct timeb *tp);`

15014

15015 DESCRIPTION

15016 The *ftime()* function shall set the *time* and *millitm* members of the **timeb** structure pointed to by
15017 *tp* to contain the seconds and milliseconds portions, respectively, of the current time in seconds
15018 since the Epoch. The contents of the *timezone* and *dstflag* members of *tp* after a call to *ftime()* are
15019 unspecified.

15020 The system clock need not have millisecond granularity. Depending on any granularity
15021 (particularly a granularity of one) renders code non-portable.

15022 RETURN VALUE

15023 Upon successful completion, the *ftime()* function shall return 0; otherwise, -1 shall be returned.

15024 ERRORS

15025 No errors are defined.

15026 EXAMPLES

15027 Getting the Current Time and Date

15028 The following example shows how to get the current system time values using the *ftime()*
15029 function. The **timeb** structure pointed to by *tp* is filled with the current system time values for
15030 *time* and *millitm*.

15031 `#include <sys/timeb.h>`

15032 `struct timeb tp;`

15033 `int status;`

15034 `...`

15035 `status = ftime(&tp);`

15036 APPLICATION USAGE

15037 For applications portability, the *time()* function should be used to determine the current time
15038 instead of *ftime()*. Realtime applications should use *clock_gettime()* to determine the current
15039 time instead of *ftime()*.

15040 RATIONALE

15041 None.

15042 FUTURE DIRECTIONS

15043 This function may be withdrawn in a future version.

15044 SEE ALSO

15045 *clock_getres()*, *ctime()*, *gettimeofday()*, *time()*, the Base Definitions volume of
15046 IEEE Std 1003.1-2001, **<sys/timeb.h>**

15047 CHANGE HISTORY

15048 First released in Issue 4, Version 2.

15049 **Issue 5**

15050 Moved from X/OPEN UNIX extension to BASE.

15051 Normative text previously in the APPLICATION USAGE section is moved to the
15052 DESCRIPTION.

15053 **Issue 6**

15054 This function is marked LEGACY.

15055 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since
15056 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*
15057 functions.

15058 **NAME**

15059 ftok — generate an IPC key

15060 **SYNOPSIS**

15061 xSI #include <sys/ipc.h>

15062 key_t ftok(const char *path, int id);

15063

15064 **DESCRIPTION**

15065 The *ftok()* function shall return a key based on *path* and *id* that is usable in subsequent calls to
 15066 *msgget()*, *semget()*, and *shmget()*. The application shall ensure that the *path* argument is the
 15067 pathname of an existing file that the process is able to *stat()*.

15068 The *ftok()* function shall return the same key value for all paths that name the same file, when
 15069 called with the same *id* value, and return different key values when called with different *id*
 15070 values or with paths that name different files existing on the same file system at the same time. It
 15071 is unspecified whether *ftok()* shall return the same key value when called again after the file
 15072 named by *path* is removed and recreated with the same name.

15073 Only the low-order 8-bits of *id* are significant. The behavior of *ftok()* is unspecified if these bits
 15074 are 0.

15075 **RETURN VALUE**

15076 Upon successful completion, *ftok()* shall return a key. Otherwise, *ftok()* shall return **(key_t)−1**
 15077 and set *errno* to indicate the error.

15078 **ERRORS**15079 The *ftok()* function shall fail if:

15080 [EACCES] Search permission is denied for a component of the path prefix.

15081 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 15082 argument.

15083 [ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname
 15084 component is longer than {NAME_MAX}.

15086 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

15087 [ENOTDIR] A component of the path prefix is not a directory.

15088 The *ftok()* function may fail if:

15089 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 15090 resolution of the *path* argument.

15091 [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result
 15092 whose length exceeds {PATH_MAX}.

15094 **EXAMPLES**15095 **Getting an IPC Key**

15096 The following example gets a unique key that can be used by the IPC functions *semget()*,
 15097 *msgget()*, and *shmget()*. The key returned by *ftok()* for this example is based on the ID value *S*
 15098 and the pathname */tmp*.

```
15099 #include <sys/ipc.h>
15100 ...
15101 key_t key;
15102 char *path = "/tmp";
15103 int id = 'S';
15104 key = ftok(path, id);
```

15105 **Saving an IPC Key**

15106 The following example gets a unique key based on the pathname */tmp* and the ID value *a*. It
 15107 also assigns the value of the resulting key to the *semkey* variable so that it will be available to a
 15108 later call to *semget()*, *msgget()*, or *shmget()*.

```
15109 #include <sys/ipc.h>
15110 ...
15111 key_t semkey;
15112 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
15113     perror("IPC error: ftok"); exit(1);
15114 }
```

15115 **APPLICATION USAGE**

15116 For maximum portability, *id* should be a single-byte character.

15117 **RATIONALE**

15118 None.

15119 **FUTURE DIRECTIONS**

15120 None.

15121 **SEE ALSO**

15122 *msgget()*, *semget()*, *shmget()*, the Base Definitions volume of IEEE Std 1003.1-2001, *<sys/ipc.h>*

15123 **CHANGE HISTORY**

15124 First released in Issue 4, Version 2.

15125 **Issue 5**

15126 Moved from X/OPEN UNIX extension to BASE.

15127 **Issue 6**

15128 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15129 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 15130 [ELOOP] error condition is added.

15131 **NAME**

15132 ftruncate — truncate a file to a specified length

15133 **SYNOPSIS**

15134 #include <unistd.h>

15135 int ftruncate(int *fildes*, off_t *length*);15136 **DESCRIPTION**15137 If *fildes* is not a valid file descriptor open for writing, the *ftruncate()* function shall fail.

15138 If *fildes* refers to a regular file, the *ftruncate()* function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate()* shall either increase the size of the file or fail. XSI-conformant systems shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate()*.

15144 Upon successful completion, if *fildes* refers to a regular file, the *ftruncate()* function shall mark for update the *st_ctime* and *st_mtime* fields of the file and the S_ISUID and S_ISGID bits of the file mode may be cleared. If the *ftruncate()* function is unsuccessful, the file is unaffected.

15147 XSI If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

15149 If *fildes* refers to a directory, *ftruncate()* shall fail.15150 If *fildes* refers to any other file type, except a shared memory object, the result is unspecified.

15151 SHM If *fildes* refers to a shared memory object, *ftruncate()* shall set the size of the shared memory object to *length*.

15153 MF|SHM If the effect of *ftruncate()* is to decrease the size of a shared memory object or memory mapped file and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

15156 MPR If the Memory Protection option is supported, references to discarded pages shall result in the generation of a SIGBUS signal; otherwise, the result of such references is undefined.

15158 MF|SHM If the effect of *ftruncate()* is to increase the size of a shared memory object, it is unspecified whether the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

15161 **RETURN VALUE**

15162 Upon successful completion, *ftruncate()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

15164 **ERRORS**15165 The *ftruncate()* function shall fail if:

15166 [EINTR] A signal was caught during execution.

15167 [EINVAL] The *length* argument was less than 0.

15168 [EFBIG] or [EINVAL]

15169 The *length* argument was greater than the maximum file size.

15170 XSI [EFBIG] The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *fildes*.

15172 [EIO] An I/O error occurred while reading from or writing to a file system.

15173 [EBADF] or [EINVAL]
 15174 The *filides* argument is not a file descriptor open for writing.
 15175 [EINVAL] The *filides* argument references a file that was opened without write
 15176 permission.
 15177 [EROFS] The named file resides on a read-only file system.

15178 EXAMPLES

15179 None.

15180 APPLICATION USAGE

15181 None.

15182 RATIONALE

15183 The *ftruncate()* function is part of IEEE Std 1003.1-2001 as it was deemed to be more useful than
 15184 *truncate()*. The *truncate()* function is provided as an XSI extension.

15185 FUTURE DIRECTIONS

15186 None.

15187 SEE ALSO

15188 *open()*, *truncate()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

15189 CHANGE HISTORY

15190 First released in Issue 4, Version 2.

15191 Issue 5

15192 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate()* in the POSIX
 15193 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is
 15194 added to the list of mandatory errors that can be returned by *ftruncate()*.

15195 Large File Summit extensions are added.

15196 Issue 6

15197 The *truncate()* function is split out into a separate reference page.

15198 The following new requirements on POSIX implementations derive from alignment with the
 15199 Single UNIX Specification:

- 15200 • The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a
 15201 regular file, the S_ISUID and S_ISGID bits in the file mode may be cleared.

15202 The following changes were made to align with the IEEE P1003.1a draft standard:

- 15203 • The DESCRIPTION text is updated.

15204 XSI-conformant systems are required to increase the size of the file if the file was previously
 15205 smaller than the size requested.

15206 **NAME**

15207 ftrylockfile — stdio locking functions

15208 **SYNOPSIS**

15209 TSF #include <stdio.h>

15210 int ftrylockfile(FILE *file);

15211

15212 **DESCRIPTION**

15213 Refer to *flockfile()*.

15214 **NAME**

15215 ftw — traverse (walk) a file tree

15216 **SYNOPSIS**15217 xSI `#include <ftw.h>`

```
15218 int ftw(const char *path, int (*fn)(const char *,
15219     const struct stat *ptr, int flag), int ndirs);
```

15220

15221 **DESCRIPTION**

15222 The *ftw()* function shall recursively descend the directory hierarchy rooted in *path*. For each
 15223 object in the hierarchy, *ftw()* shall call the function pointed to by *fn*, passing it a pointer to a
 15224 null-terminated character string containing the name of the object, a pointer to a **stat** structure
 15225 containing information about the object, and an integer. Possible values of the integer, defined
 15226 in the **<ftw.h>** header, are:

15227 FTW_D For a directory.

15228 FTW_DNR For a directory that cannot be read.

15229 FTW_F For a file.

15230 FTW_SL For a symbolic link (but see also FTW_NS below).

15231 FTW_NS For an object other than a symbolic link on which *stat()* could not successfully be
 15232 executed. If the object is a symbolic link and *stat()* failed, it is unspecified whether
 15233 *ftw()* passes FTW_SL or FTW_NS to the user-supplied function.

15234 If the integer is FTW_DNR, descendants of that directory shall not be processed. If the integer is
 15235 FTW_NS, the **stat** structure contains undefined values. An example of an object that would
 15236 cause FTW_NS to be passed to the function pointed to by *fn* would be a file in a directory with
 15237 read but without execute (search) permission.

15238 The *ftw()* function shall visit a directory before visiting any of its descendants.15239 The *ftw()* function shall use at most one file descriptor for each level in the tree.15240 The argument *ndirs* should be in the range [1,{OPEN_MAX}].

15241 The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a
 15242 non-zero value, or some error, other than [EACCES], is detected within *ftw()*.

15243 The *ndirs* argument shall specify the maximum number of directory streams or file descriptors
 15244 or both available for use by *ftw()* while traversing the tree. When *ftw()* returns it shall close any
 15245 directory streams and file descriptors it uses not counting any opened by the application-
 15246 supplied *fn* function.

15247 The results are unspecified if the application-supplied *fn* function does not preserve the current
 15248 working directory.

15249 The *ftw()* function need not be reentrant. A function that is not required to be reentrant is not
 15250 required to be thread-safe.

15251 **RETURN VALUE**

15252 If the tree is exhausted, *ftw()* shall return 0. If the function pointed to by *fn* returns a non-zero
 15253 value, *ftw()* shall stop its tree traversal and return whatever value was returned by the function
 15254 pointed to by *fn*. If *ftw()* detects an error, it shall return -1 and set *errno* to indicate the error.

15255 If *ftw()* encounters an error other than [EACCES] (see FTW_DNR and FTW_NS above), it shall
 15256 return -1 and set *errno* to indicate the error. The external variable *errno* may contain any error

15257 value that is possible when a directory is opened or when one of the *stat* functions is executed on
 15258 a directory or file.

15259 ERRORS

15260 The *ftw()* function shall fail if:

15261 [EACCES] Search permission is denied for any component of *path* or read permission is
 15262 denied for *path*.

15263 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 15264 argument.

15265 [ENAMETOOLONG]
 15266 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 15267 component is longer than {NAME_MAX}.

15268 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

15269 [ENOTDIR] A component of *path* is not a directory.

15270 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current
 15271 programming environment for one or more files found in the file hierarchy.

15272 The *ftw()* function may fail if:

15273 [EINVAL] The value of the *ndirs* argument is invalid.

15274 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 15275 resolution of the *path* argument.

15276 [ENAMETOOLONG]
 15277 Pathname resolution of a symbolic link produced an intermediate result
 15278 whose length exceeds {PATH_MAX}.

15279 In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set
 15280 accordingly.

15281 EXAMPLES

15282 Walking a Directory Structure

15283 The following example walks the current directory structure, calling the *fn* function for every
 15284 directory entry, using at most 10 file descriptors:

```
15285 #include <ftw.h>
15286 ...
15287 if (ftw(".", fn, 10) != 0) {
15288     perror("ftw"); exit(2);
15289 }
```

15290 APPLICATION USAGE

15291 The *ftw()* function may allocate dynamic storage during its operation. If *ftw()* is forcibly
 15292 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*
 15293 or an interrupt routine, *ftw()* does not have a chance to free that storage, so it remains
 15294 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has
 15295 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next
 15296 invocation.

15297 **RATIONALE**

15298 None.

15299 **FUTURE DIRECTIONS**

15300 None.

15301 **SEE ALSO**

15302 *longjmp()*, *lstat()*, *malloc()*, *nftw()*, *opendir()*, *siglongjmp()*, *stat()*, the Base Definitions volume of
15303 IEEE Std 1003.1-2001, <**ftw.h**>, <**sys/stat.h**>

15304 **CHANGE HISTORY**

15305 First released in Issue 1. Derived from Issue 1 of the SVID.

15306 **Issue 5**

15307 UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections are changed to EX.

15308 **Issue 6**

15309 The ERRORS section is updated as follows:

- 15310 • The wording of the mandatory [ELOOP] error condition is updated.
- 15311 • A second optional [ELOOP] error condition is added.
- 15312 • The [EOVERFLOW] mandatory error condition is added.

15313 Text is added to the DESCRIPTION to say that the *ftw()* function need not be reentrant and that
15314 the results are unspecified if the application-supplied *fn* function does not preserve the current
15315 working directory.

15316 NAME

15317 funlockfile — stdio locking functions

15318 SYNOPSIS

15319 TSF `#include <stdio.h>`

15320 `void funlockfile(FILE *file);`

15321

15322 DESCRIPTION

15323 Refer to *flockfile()*.

15324 **NAME**

15325 fwide — set stream orientation

15326 **SYNOPSIS**

15327 #include <stdio.h>

15328 #include <wchar.h>

15329 int fwide(FILE **stream*, int *mode*);15330 **DESCRIPTION**

15331 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 15332 conflict between the requirements described here and the ISO C standard is unintentional. This
 15333 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15334 The *fwide()* function shall determine the orientation of the stream pointed to by *stream*. If *mode* is
 15335 greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less
 15336 than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero
 15337 and the function does not alter the orientation of the stream.

15338 If the orientation of the stream has already been determined, *fwide()* shall not change it.

15339 CX Since no return value is reserved to indicate an error, an application wishing to check for error
 15340 situations should set *errno* to 0, then call *fwide()*, then check *errno*, and if it is non-zero, assume
 15341 an error has occurred.

15342 **RETURN VALUE**

15343 The *fwide()* function shall return a value greater than zero if, after the call, the stream has wide-
 15344 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no
 15345 orientation.

15346 **ERRORS**15347 The *fwide()* function may fail if:

15348 CX [EBADF] The *stream* argument is not a valid stream.

15349 **EXAMPLES**

15350 None.

15351 **APPLICATION USAGE**

15352 A call to *fwide()* with *mode* set to zero can be used to determine the current orientation of a
 15353 stream.

15354 **RATIONALE**

15355 None.

15356 **FUTURE DIRECTIONS**

15357 None.

15358 **SEE ALSO**15359 The Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>15360 **CHANGE HISTORY**

15361 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 15362 (E).

15363 **Issue 6**

15364 Extensions beyond the ISO C standard are marked.

15365 **NAME**

15366 fwprintf, swprintf, wprintf — print formatted wide-character output

15367 **SYNOPSIS**

15368 #include <stdio.h>

15369 #include <wchar.h>

15370 int fwprintf(FILE *restrict *stream*, const wchar_t *restrict *format*, ...);15371 int swprintf(wchar_t *restrict *ws*, size_t *n*,15372 const wchar_t *restrict *format*, ...);15373 int wprintf(const wchar_t *restrict *format*, ...);15374 **DESCRIPTION**

15375 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 15376 conflict between the requirements described here and the ISO C standard is unintentional. This
 15377 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15378 The *fwprintf()* function shall place output on the named output *stream*. The *wprintf()* function
 15379 shall place output on the standard output stream *stdout*. The *swprintf()* function shall place
 15380 output followed by the null wide character in consecutive wide characters starting at **ws*; no
 15381 more than *n* wide characters shall be written, including a terminating null wide character, which
 15382 is always added (unless *n* is zero).

15383 Each of these functions shall convert, format, and print its arguments under control of the *format*
 15384 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-*
 15385 *characters*, which are simply copied to the output stream, and *conversion specifications*, each of
 15386 which results in the fetching of zero or more arguments. The results are undefined if there are
 15387 insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the
 15388 excess arguments are evaluated but are otherwise ignored.

15389 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 15390 to the next unused argument. In this case, the conversion specifier wide character % (see below)
 15391 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}],
 15392 giving the position of the argument in the argument list. This feature provides for the definition
 15393 of *format* wide-character strings that select arguments in an order appropriate to specific
 15394 languages (see the EXAMPLES section).

15395 The *format* can contain either numbered argument specifications (that is, "%n\$" and "%m\$"), or
 15396 unnumbered argument conversion specifications (that is, % and *), but not both. The only
 15397 exception to this is that %% can be mixed with the "%n\$" form. The results of mixing numbered
 15398 and unnumbered argument specifications in a *format* wide-character string are undefined. When
 15399 numbered argument specifications are used, specifying the *N*th argument requires that all the
 15400 leading arguments, from the first to the (*N*−1)th, are specified in the *format* wide-character string.

15401 In *format* wide-character strings containing the "%n\$" form of conversion specification,
 15402 numbered arguments in the argument list can be referenced from the *format* wide-character
 15403 string as many times as required.

15404 In *format* wide-character strings containing the % form of conversion specification, each
 15405 argument in the argument list shall be used exactly once.

15406 CX All forms of the *fwprintf()* function allow for the insertion of a locale-dependent radix character
 15407 in the output string, output as a wide-character value. The radix character is defined in the
 15408 program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix
 15409 character is not defined, the radix character shall default to a period ('.').

15410 XSI Each conversion specification is introduced by the '%' wide character or by the wide-character
 15411 sequence "%n\$", after which the following appear in sequence:

- 15412 • Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- 15413 • An optional minimum *field width*. If the converted value has fewer wide characters than the
- 15414 field width, it shall be padded with spaces by default on the left; it shall be padded on the
- 15415 right, if the left-adjustment flag ('-'), described below, is given to the field width. The field
- 15416 width takes the form of an asterisk ('*'), described below, or a decimal integer.
- 15417 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x,
- 15418 and X conversion specifiers; the number of digits to appear after the radix character for the a,
- 15419 A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g
- 15420 and G conversion specifiers; or the maximum number of wide characters to be printed from a
- 15421 string in the s conversion specifiers. The precision takes the form of a period ('.') followed
- 15422 either by an asterisk ('*'), described below, or an optional decimal digit string, where a null
- 15423 digit string is treated as 0. If a precision appears with any other conversion wide character,
- 15424 the behavior is undefined.
- 15425 • An optional length modifier that specifies the size of the argument.
- 15426 • A *conversion specifier* wide character that indicates the type of conversion to be applied.
- 15427 A field width, or precision, or both, may be indicated by an asterisk ('*'). In this case an
- 15428 argument of type **int** supplies the field width or precision. Applications shall ensure that
- 15429 arguments specifying field width, or precision, or both appear in that order before the argument,
- 15430 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field
- 15431 XSI width. A negative precision is taken as if the precision were omitted. In *format* wide-character
- 15432 strings containing the "%n\$" form of a conversion specification, a field width or precision may
- 15433 be indicated by the sequence "%m\$", where *m* is a decimal integer in the range
- 15434 [1,{NL_ARGMAX}] giving the position in the argument list (after the *format* argument) of an
- 15435 integer argument containing the field width or precision, for example:
- 15436 `wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);`
- 15437 The flag wide characters and their meanings are:
- 15438 XSI ' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G)
- 15439 shall be formatted with thousands' grouping wide characters. For other conversions,
- 15440 the behavior is undefined. The numeric grouping wide character is used.
- 15441 - The result of the conversion shall be left-justified within the field. The conversion shall
- 15442 be right-justified if this flag is not specified.
- 15443 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The
- 15444 conversion shall begin with a sign only when a negative value is converted if this flag is
- 15445 not specified.
- 15446 <space> If the first wide character of a signed conversion is not a sign, or if a signed conversion
- 15447 results in no wide characters, a <space> shall be prefixed to the result. This means that
- 15448 if the <space> and '+' flags both appear, the <space> flag shall be ignored.
- 15449 # Specifies that the value is to be converted to an alternative form. For o conversion, it
- 15450 increases the precision (if necessary) to force the first digit of the result to be 0. For x or
- 15451 X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e,
- 15452 E, f, F, g, and G conversion specifiers, the result shall always contain a radix character,
- 15453 even if no digits follow it. Without this flag, a radix character appears in the result of
- 15454 these conversions only if a digit follows it. For g and G conversion specifiers, trailing
- 15455 zeros shall *not* be removed from the result as they normally are. For other conversion
- 15456 specifiers, the behavior is undefined.

15457 0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros
 15458 (following any indication of sign or base) are used to pad to the field width; no space
 15459 padding is performed. If the '0' and '-' flags both appear, the '0' flag shall be
 15460 ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0'
 15461 flag shall be ignored. If the '0' and '-' flags both appear, the grouping wide
 15462 characters are inserted before zero padding. For other conversions, the behavior is
 15463 undefined.

15464 The length modifiers and their meanings are:

15465 hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char**
 15466 or **unsigned char** argument (the argument will have been promoted according to the
 15467 integer promotions, but its value shall be converted to **signed char** or **unsigned char**
 15468 before printing); or that a following n conversion specifier applies to a pointer to a
 15469 **signed char** argument.

15470 h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or
 15471 **unsigned short** argument (the argument will have been promoted according to the
 15472 integer promotions, but its value shall be converted to **short** or **unsigned short** before
 15473 printing); or that a following n conversion specifier applies to a pointer to a **short**
 15474 argument.

15475 l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or
 15476 **unsigned long** argument; that a following n conversion specifier applies to a pointer to
 15477 a **long** argument; that a following c conversion specifier applies to a **wint_t** argument;
 15478 that a following s conversion specifier applies to a pointer to a **wchar_t** argument; or
 15479 has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.

15480 ll (ell-ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long** or
 15481 **unsigned long long** argument; or that a following n conversion specifier applies to a
 15482 pointer to a **long long** argument.

15484 j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax_t**
 15485 or **uintmax_t** argument; or that a following n conversion specifier applies to a pointer
 15486 to an **intmax_t** argument.

15487 z Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **size_t** or the
 15488 corresponding signed integer type argument; or that a following n conversion specifier
 15489 applies to a pointer to a signed integer type corresponding to a **size_t** argument.

15490 t Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **ptrdiff_t** or
 15491 the corresponding **unsigned** type argument; or that a following n conversion specifier
 15492 applies to a pointer to a **ptrdiff_t** argument.

15493 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a **long**
 15494 **double** argument.

15495 If a length modifier appears with any conversion specifier other than as specified above, the
 15496 behavior is undefined.

15497 The conversion specifiers and their meanings are:

15498 d, i The **int** argument shall be converted to a signed decimal in the style "[-] dddd". The
 15499 precision specifies the minimum number of digits to appear; if the value being
 15500 converted can be represented in fewer digits, it shall be expanded with leading zeros.
 15501 The default precision shall be 1. The result of converting zero with an explicit precision
 15502 of zero shall be no wide characters.

15503	o	The unsigned argument shall be converted to unsigned octal format in the style "dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
15504		
15505		
15506		
15507		
15508	u	The unsigned argument shall be converted to unsigned decimal format in the style "dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
15509		
15510		
15511		
15512		
15513	x	The unsigned argument shall be converted to unsigned hexadecimal format in the style "dddd"; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
15514		
15515		
15516		
15517		
15518	X	Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".
15519		
15520	f, F	The double argument shall be converted to decimal notation in the style "[-] ddd.ddd", where the number of digits after the radix character shall be equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix character appears, at least one digit shall appear before it. The value shall be rounded in an implementation-defined manner to the appropriate number of digits.
15521		
15522		
15523		
15524		
15525		
15526		A double argument representing an infinity shall be converted in one of the styles "[-]inf" or "[-]infinity"; which style is implementation-defined. A double argument representing a NaN shall be converted in one of the styles "[-]nan" or "[-]nan(<i>n-char-sequence</i>)"; which style, and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The F conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.
15527		
15528		
15529		
15530		
15531		
15532	e, E	The double argument shall be converted in the style "[-] d.ddde±dd", where there shall be one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it shall be equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall appear. The value shall be rounded in an implementation-defined manner to the appropriate number of digits. The E conversion wide character shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.
15533		
15534		
15535		
15536		
15537		
15538		
15539		
15540		A double argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.
15541		
15542	g, G	The double argument shall be converted in the style f or e (or in the style F or E in the case of a G conversion specifier), with the precision specifying the number of significant digits. If an explicit precision is zero, it shall be taken as 1. The style used depends on the value converted; style e (or E) shall be used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros shall be removed from the fractional portion of the result; a radix character shall appear only if it is followed by a digit.
15543		
15544		
15545		
15546		
15547		
15548		
15549		A double argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.
15550		

15551	a, A	A double argument representing a floating-point number shall be converted in the style "[-] 0xh.hhhhp±d", where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type double , except that trailing zeros may be omitted; if the precision is zero and the '#' flag is not specified, no decimal-point wide character shall appear. The letters "abcdef" are used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.
15552		
15553		
15554		
15555		
15556		
15557		
15558		
15559		
15560		
15561		
15562		
15563		
15564		
15565		A double argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.
15566		
15567	c	If no l (ell) qualifier is present, the int argument shall be converted to a wide character as if by calling the <i>btowc()</i> function and the resulting wide character shall be written. Otherwise, the wint_t argument shall be converted to wchar_t , and written.
15568		
15569		
15570	s	If no l (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the <i>mbrtowc()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.
15571		
15572		
15573		
15574		
15575		
15576		
15577		
15578		
15579		If an l (ell) qualifier is present, the application shall ensure that the argument is a pointer to an array of type wchar_t . Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.
15580		
15581		
15582		
15583		
15584		
15585	p	The application shall ensure that the argument is a pointer to void . The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.
15586		
15587		
15588	n	The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the <i>fwprintf()</i> functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
15589		
15590		
15591		
15592		
15593	XSI C	Equivalent to lc.
15594	XSI S	Equivalent to ls.
15595	%	Output a '%' wide character; no argument shall be converted. The entire conversion specification shall be %%.
15596		

- 15597 If a conversion specification does not match one of the above forms, the behavior is undefined.
- 15598 In no case does a nonexistent or small field width cause truncation of a field; if the result of a
15599 conversion is wider than the field width, the field shall be expanded to contain the conversion
15600 result. Characters generated by *fwprintf()* and *wprintf()* shall be printed as if *fputwc()* had been
15601 called.
- 15602 For a and A conversions, if FLT_RADIX is not a power of 2 and the result is not exactly
15603 representable in the given precision, the result should be one of the two adjacent numbers in
15604 hexadecimal floating style with the given precision, with the extra stipulation that the error
15605 should have a correct sign for the current rounding direction.
- 15606 For e, E, f, F, g, and G conversion specifiers, if the number of significant decimal digits is at most
15607 DECIMAL_DIG, then the result should be correctly rounded. If the number of significant
15608 decimal digits is more than DECIMAL_DIG but the source value is exactly representable with
15609 DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros.
15610 Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having
15611 DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L \leq$
15612 $D \leq U$, with the extra stipulation that the error should have a correct sign for the current
15613 rounding direction.
- 15614 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the call to a
15615 successful execution of *fwprintf()* or *wprintf()* and the next successful completion of a call to
15616 *fflush()* or *fclose()* on the same stream, or a call to *exit()* or *abort()*.
- 15617 **RETURN VALUE**
- 15618 Upon successful completion, these functions shall return the number of wide characters
15619 transmitted, excluding the terminating null wide character in the case of *swprintf()*, or a negative
15620 CX value if an output error was encountered, and set *errno* to indicate the error.
- 15621 If *n* or more wide characters were requested to be written, *swprintf()* shall return a negative
15622 CX value, and set *errno* to indicate the error.
- 15623 **ERRORS**
- 15624 For the conditions under which *fwprintf()* and *wprintf()* fail and may fail, refer to *fputwc()*.
- 15625 In addition, all forms of *fwprintf()* may fail if:
- 15626 XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been
15627 detected.
- 15628 XSI [EINVAL] There are insufficient arguments.
- 15629 In addition, *wprintf()* and *fwprintf()* may fail if:
- 15630 XSI [ENOMEM] Insufficient storage space is available.
- 15631 **EXAMPLES**
- 15632 To print the language-independent date and time format, the following statement could be used:
- 15633 `wprintf(format, weekday, month, day, hour, min);`
- 15634 For American usage, *format* could be a pointer to the wide-character string:
- 15635 `L"%s, %s %d, %d:%.2d\n"`
- 15636 producing the message:
- 15637 `Sunday, July 3, 10:02`
- 15638 whereas for German usage, *format* could be a pointer to the wide-character string:

15639 L"%1\$s, %3\$d. %2\$s, %4\$d:%5\$.2d\n"

15640 producing the message:

15641 Sonntag, 3. Juli, 10:02

15642 APPLICATION USAGE

15643 None.

15644 RATIONALE

15645 None.

15646 FUTURE DIRECTIONS

15647 None.

15648 SEE ALSO

15649 *btowc()*, *fputwc()*, *fwscanf()*, *mbrtowc()*, *setlocale()*, the Base Definitions volume of
15650 IEEE Std 1003.1-2001, Chapter 7, Locale, <stdio.h>, <wchar.h>

15651 CHANGE HISTORY

15652 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
15653 (E).

15654 Issue 6

15655 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing
15656 the case if *n* or more wide characters are requested to be written using *swprintf()*.

15657 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15658 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15659 • The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.
- 15660 • The DESCRIPTION is updated.
- 15661 • The hh, ll, j, t, and z length modifiers are added.
- 15662 • The a, A, and F conversion characters are added.
- 15663 • XSI shading is removed from the description of character string representations of infinity
15664 and NaN floating-point values.

15665 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
15666 specification” consistently.

15667 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

15668 **NAME**

15669 fwrite — binary output

15670 **SYNOPSIS**

15671 #include <stdio.h>

15672 size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
 15673 FILE *restrict stream);

15674 **DESCRIPTION**

15675 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 15676 conflict between the requirements described here and the ISO C standard is unintentional. This
 15677 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15678 The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose
 15679 size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be
 15680 made to the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly
 15681 overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by
 15682 the number of bytes successfully written. If an error occurs, the resulting value of the file-
 15683 position indicator for the stream is unspecified.

15684 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
 15685 execution of *fwrite()* and the next successful completion of a call to *fflush()* or *fclose()* on the
 15686 same stream, or a call to *exit()* or *abort()*.

15687 **RETURN VALUE**

15688 The *fwrite()* function shall return the number of elements successfully written, which may be
 15689 less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0 and the
 15690 state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for
 15691 CX the stream shall be set, and *errno* shall be set to indicate the error.

15692 **ERRORS**15693 Refer to *fputc()*.15694 **EXAMPLES**

15695 None.

15696 **APPLICATION USAGE**

15697 Because of possible differences in element length and byte ordering, files written using *fwrite()*
 15698 are application-dependent, and possibly cannot be read using *fread()* by a different application
 15699 or by the same application on a different processor.

15700 **RATIONALE**

15701 None.

15702 **FUTURE DIRECTIONS**

15703 None.

15704 **SEE ALSO**

15705 *ferror()*, *fopen()*, *printf()*, *putc()*, *puts()*, *write()*, the Base Definitions volume of
 15706 IEEE Std 1003.1-2001, <**stdio.h**>

15707 **CHANGE HISTORY**

15708 First released in Issue 1. Derived from Issue 1 of the SVID.

15709 **Issue 6**

15710 Extensions beyond the ISO C standard are marked.

15711 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

15712

- The *fwrite()* prototype is updated.

15713

- The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.

15714 **NAME**

15715 fwscanf, swscanf, wscanf — convert formatted wide-character input

15716 **SYNOPSIS**

15717 #include <stdio.h>

15718 #include <wchar.h>

15719 int fwscanf(FILE *restrict *stream*, const wchar_t *restrict *format*, ...);15720 int swscanf(const wchar_t *restrict *ws*,15721 const wchar_t *restrict *format*, ...);15722 int wscanf(const wchar_t *restrict *format*, ...);15723 **DESCRIPTION**

15724 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 15725 conflict between the requirements described here and the ISO C standard is unintentional. This
 15726 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15727 The *fwscanf()* function shall read from the named input *stream*. The *wscanf()* function shall read
 15728 from the standard input stream *stdin*. The *swscanf()* function shall read from the wide-character
 15729 string *ws*. Each function reads wide characters, interprets them according to a format, and stores
 15730 the results in its arguments. Each expects, as arguments, a control wide-character string *format*
 15731 described below, and a set of *pointer* arguments indicating where the converted input should be
 15732 stored. The result is undefined if there are insufficient arguments for the format. If the *format* is
 15733 exhausted while arguments remain, the excess arguments are evaluated but are otherwise
 15734 ignored.

15735 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 15736 to the next unused argument. In this case, the conversion specifier wide character % (see below)
 15737 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}].
 15738 This feature provides for the definition of *format* wide-character strings that select arguments in
 15739 an order appropriate to specific languages. In *format* wide-character strings containing the
 15740 "%n\$" form of conversion specifications, it is unspecified whether numbered arguments in the
 15741 argument list can be referenced from the *format* wide-character string more than once.

15742 The *format* can contain either form of a conversion specification—that is, % or "%n\$" — but the
 15743 two forms cannot normally be mixed within a single *format* wide-character string. The only
 15744 exception to this is that %% or %* can be mixed with the "%n\$" form. When numbered
 15745 argument specifications are used, specifying the *N*th argument requires that all the leading
 15746 arguments, from the first to the (*N*−1)th, are pointers.

15747 CX The *fwscanf()* function in all its forms allows for detection of a language-dependent radix
 15748 character in the input string, encoded as a wide-character value. The radix character is defined in
 15749 the program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the
 15750 radix character is not defined, the radix character shall default to a period (' . ').

15751 The *format* is a wide-character string composed of zero or more directives. Each directive is
 15752 composed of one of the following: one or more white-space wide characters (<space>s, <tab>s,
 15753 <newline>s, <vertical-tab>s, or <form-feed>s); an ordinary wide character (neither '%' nor a
 15754 white-space character); or a conversion specification. Each conversion specification is introduced
 15755 XSI by a '%' or the sequence "%n\$" after which the following appear in sequence:

- 15756 • An optional assignment-suppressing character ' * '.
- 15757 • An optional non-zero decimal integer that specifies the maximum field width.
- 15758 • An optional length modifier that specifies the size of the receiving object.

15759 • A conversion specifier wide character that specifies the type of conversion to be applied. The
15760 valid conversion specifiers are described below.

15761 The *fwscanf()* functions shall execute each directive of the format in turn. If a directive fails, as
15762 detailed below, the function shall return. Failures are described as input failures (due to the
15763 unavailability of input bytes) or matching failures (due to inappropriate input).

15764 A directive composed of one or more white-space wide characters is executed by reading input
15765 until no more valid input can be read, or up to the first wide character which is not a white-
15766 space wide character, which remains unread.

15767 A directive that is an ordinary wide character shall be executed as follows. The next wide
15768 character is read from the input and compared with the wide character that comprises the
15769 directive; if the comparison shows that they are not equivalent, the directive shall fail, and the
15770 differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding
15771 error, or a read error prevents a wide character from being read, the directive shall fail.

15772 A directive that is a conversion specification defines a set of matching input sequences, as
15773 described below for each conversion wide character. A conversion specification is executed in
15774 the following steps.

15775 Input white-space wide characters (as specified by *iswspace()*) shall be skipped, unless the
15776 conversion specification includes a *[], c,* or *n* conversion specifier.

15777 An item shall be read from the input, unless the conversion specification includes an *n*
15778 conversion specifier wide character. An input item is defined as the longest sequence of input
15779 wide characters, not exceeding any specified field width, which is an initial subsequence of a
15780 matching sequence. The first wide character, if any, after the input item shall remain unread. If
15781 the length of the input item is zero, the execution of the conversion specification shall fail; this
15782 condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented
15783 input from the stream, in which case it is an input failure.

15784 Except in the case of a *%* conversion specifier, the input item (or, in the case of a *%n* conversion
15785 specification, the count of input wide characters) shall be converted to a type appropriate to the
15786 conversion wide character. If the input item is not a matching sequence, the execution of the
15787 conversion specification shall fail; this condition is a matching failure. Unless assignment
15788 suppression was indicated by a *'*'*, the result of the conversion shall be placed in the object
15789 pointed to by the first argument following the *format* argument that has not already received a
15790 xsi conversion result if the conversion specification is introduced by *%*, or in the *n*th argument if
15791 introduced by the wide-character sequence *"%n\$"*. If this object does not have an appropriate
15792 type, or if the result of the conversion cannot be represented in the space provided, the behavior
15793 is undefined.

15794 The length modifiers and their meanings are:

15795 hh	Specifies that a following <i>d, i, o, u, x, X,</i> or <i>n</i> conversion specifier applies to an 15796 argument with type pointer to signed char or unsigned char .
15797 h	Specifies that a following <i>d, i, o, u, x, X,</i> or <i>n</i> conversion specifier applies to an 15798 argument with type pointer to short or unsigned short .
15799 l (ell)	Specifies that a following <i>d, i, o, u, x, X,</i> or <i>n</i> conversion specifier applies to an 15800 argument with type pointer to long or unsigned long ; that a following <i>a, A, e, E, f, F, g,</i> 15801 or <i>G</i> conversion specifier applies to an argument with type pointer to double ; or that a 15802 following <i>c, s,</i> or <i>[]</i> conversion specifier applies to an argument with type pointer to 15803 wchar_t .

15804	ll (ell-ell)	
15805		Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
15806		argument with type pointer to long long or unsigned long long .
15807	j	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
15808		argument with type pointer to intmax_t or uintmax_t .
15809	z	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
15810		argument with type pointer to size_t or the corresponding signed integer type.
15811	t	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
15812		argument with type pointer to ptrdiff_t or the corresponding unsigned type.
15813	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an
15814		argument with type pointer to long double .
15815		If a length modifier appears with any conversion specifier other than as specified above, the
15816		behavior is undefined.
15817		The following conversion specifier wide characters are valid:
15818	d	Matches an optionally signed decimal integer, whose format is the same as expected for
15819		the subject sequence of <i>wcstol()</i> with the value 10 for the <i>base</i> argument. In the absence
15820		of a size modifier, the application shall ensure that the corresponding argument is a
15821		pointer to int .
15822	i	Matches an optionally signed integer, whose format is the same as expected for the
15823		subject sequence of <i>wcstol()</i> with 0 for the <i>base</i> argument. In the absence of a size
15824		modifier, the application shall ensure that the corresponding argument is a pointer to
15825		int .
15826	o	Matches an optionally signed octal integer, whose format is the same as expected for
15827		the subject sequence of <i>wcstoul()</i> with the value 8 for the <i>base</i> argument. In the absence
15828		of a size modifier, the application shall ensure that the corresponding argument is a
15829		pointer to unsigned .
15830	u	Matches an optionally signed decimal integer, whose format is the same as expected for
15831		the subject sequence of <i>wcstoul()</i> with the value 10 for the <i>base</i> argument. In the absence
15832		of a size modifier, the application shall ensure that the corresponding argument is a
15833		pointer to unsigned .
15834	x	Matches an optionally signed hexadecimal integer, whose format is the same as
15835		expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument.
15836		In the absence of a size modifier, the application shall ensure that the corresponding
15837		argument is a pointer to unsigned .
15838	a, e, f, g	
15839		Matches an optionally signed floating-point number, infinity, or NaN whose format is
15840		the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size
15841		modifier, the application shall ensure that the corresponding argument is a pointer to
15842		float .
15843		If the <i>fwprintf()</i> family of functions generates character string representations for
15844		infinity and NaN (a symbolic entity encoded in floating-point format) to support
15845		IEEE Std 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.
15846	s	Matches a sequence of non white-space wide characters. If no l (ell) qualifier is present,
15847		characters from the input field shall be converted as if by repeated calls to the
15848		<i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object

15849		initialized to zero before the first wide character is converted. The application shall
15850		ensure that the corresponding argument is a pointer to a character array large enough
15851		to accept the sequence and the terminating null character, which shall be added
15852		automatically.
15853		Otherwise, the application shall ensure that the corresponding argument is a pointer to
15854		an array of wchar_t large enough to accept the sequence and the terminating null wide
15855		character, which shall be added automatically.
15856	[	Matches a non-empty sequence of wide characters from a set of expected wide
15857		characters (the <i>scanset</i>). If no l (ell) qualifier is present, wide characters from the input
15858		field shall be converted as if by repeated calls to the <i>wrtomb()</i> function, with the
15859		conversion state described by an mbstate_t object initialized to zero before the first
15860		wide character is converted. The application shall ensure that the corresponding
15861		argument is a pointer to a character array large enough to accept the sequence and the
15862		terminating null character, which shall be added automatically.
15863		If an l (ell) qualifier is present, the application shall ensure that the corresponding
15864		argument is a pointer to an array of wchar_t large enough to accept the sequence and
15865		the terminating null wide character, which shall be added automatically.
15866		The conversion specification includes all subsequent wide characters in the <i>format</i>
15867		string up to and including the matching right square bracket (']'). The wide
15868		characters between the square brackets (the <i>scanlist</i>) comprise the scanset, unless the
15869		wide character after the left square bracket is a circumflex ('^'), in which case the
15870		scanset contains all wide characters that do not appear in the scanlist between the
15871		circumflex and the right square bracket. If the conversion specification begins with
15872		"[]" or "[^]", the right square bracket is included in the scanlist and the next right
15873		square bracket is the matching right square bracket that ends the conversion
15874		specification; otherwise, the first right square bracket is the one that ends the
15875		conversion specification. If a '-' is in the scanlist and is not the first wide character,
15876		nor the second where the first wide character is a '^', nor the last wide character, the
15877		behavior is implementation-defined.
15878	c	Matches a sequence of wide characters of exactly the number specified by the field
15879		width (1 if no field width is present in the conversion specification).
15880		If no l (ell) length modifier is present, characters from the input field shall be converted
15881		as if by repeated calls to the <i>wrtomb()</i> function, with the conversion state described by
15882		an mbstate_t object initialized to zero before the first wide character is converted. The
15883		corresponding argument shall be a pointer to the initial element of a character array
15884		large enough to accept the sequence. No null character is added.
15885		If an l (ell) length modifier is present, the corresponding argument shall be a pointer to
15886		the initial element of an array of wchar_t large enough to accept the sequence. No null
15887		wide character is added.
15888		Otherwise, the application shall ensure that the corresponding argument is a pointer to
15889		an array of wchar_t large enough to accept the sequence. No null wide character is
15890		added.
15891	p	Matches an implementation-defined set of sequences, which shall be the same as the set
15892		of sequences that is produced by the %p conversion specification of the corresponding
15893		<i>fwprintf()</i> functions. The application shall ensure that the corresponding argument is a
15894		pointer to a pointer to void . The interpretation of the input item is implementation-
15895		defined. If the input item is a value converted earlier during the same program
15896		execution, the pointer that results shall compare equal to that value; otherwise, the

15897		behavior of the %p conversion is undefined.
15898	n	No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which is to be written the number of wide characters read from the input so far by this call to the <i>fwscanf()</i> functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing wide character or a field width, the behavior is undefined.
15899		
15900		
15901		
15902		
15903		
15904		
15905	XSI	C Equivalent to lc.
15906	XSI	S Equivalent to ls.
15907	%	Matches a single '%' wide character; no conversion or assignment shall occur. The complete conversion specification shall be %%.
15908		
15909		If a conversion specification is invalid, the behavior is undefined.
15910		The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to, respectively, a, e, f, g, and x.
15911		
15912		If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide characters matching the current conversion specification (except for %n) have been read (other than leading white-space, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.
15913		
15914		
15915		
15916		
15917		
15918		Reaching the end of the string in <i>swscanf()</i> shall be equivalent to encountering end-of-file for <i>fwscanf()</i> .
15919		
15920		If conversion terminates on a conflicting input, the offending input shall be left unread in the input. Any trailing white space (including <newline>) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the %n conversion specification.
15921		
15922		
15923		
15924		
15924	CX	The <i>fwscanf()</i> and <i>wscanf()</i> functions may mark the <i>st_atime</i> field of the file associated with <i>stream</i> for update. The <i>st_atime</i> field shall be marked for update by the first successful execution of <i>fgetc()</i> , <i>fgetwc()</i> , <i>fgets()</i> , <i>fgetws()</i> , <i>fread()</i> , <i>getc()</i> , <i>getwc()</i> , <i>getchar()</i> , <i>getwchar()</i> , <i>gets()</i> , <i>fscanf()</i> , or <i>fwscanf()</i> using <i>stream</i> that returns data not supplied by a prior call to <i>ungetc()</i> .
15925		
15926		
15927		
15928	RETURN VALUE	
15929		Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF shall be returned. If a read error occurs, the error indicator for the stream is set, EOF shall be returned, and <i>errno</i> shall be set to indicate the error.
15930		
15931		
15932		
15932	CX	
15933		
15934	ERRORS	
15935		For the conditions under which the <i>fwscanf()</i> functions shall fail and may fail, refer to <i>fgetwc()</i> .
15936		In addition, <i>fwscanf()</i> may fail if:
15937	XSI	[EILSEQ] Input byte sequence does not form a valid character.
15938	XSI	[EINVAL] There are insufficient arguments.

15939 **EXAMPLES**

15940 The call:

```
15941 int i, n; float x; char name[50];
15942 n = wscanf(L"%d%f%s", &i, &x, name);
```

15943 with the input line:

15944 25 54.32E-1 Hamster

15945 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string
 15946 "Hamster".

15947 The call:

```
15948 int i; float x; char name[50];
15949 (void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

15950 with input:

15951 56789 0123 56a72

15952 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to
 15953 *getchar()* shall return the character 'a'.

15954 **APPLICATION USAGE**

15955 In format strings containing the '%' form of conversion specifications, each argument in the
 15956 argument list is used exactly once.

15957 **RATIONALE**

15958 None.

15959 **FUTURE DIRECTIONS**

15960 None.

15961 **SEE ALSO**

15962 *getwc()*, *fwprintf()*, *setlocale()*, *wctod()*, *wctol()*, *wcstoul()*, *wcrtomb()*, the Base Definitions
 15963 volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <langinfo.h>, <stdio.h>, <wchar.h>

15964 **CHANGE HISTORY**

15965 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 15966 (E).

15967 **Issue 6**

15968 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15969 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15970 • The prototypes for *fwscanf()* and *swscanf()* are updated.
- 15971 • The DESCRIPTION is updated.
- 15972 • The hh, ll, j, t, and z length modifiers are added.
- 15973 • The a, A, and F conversion characters are added.

15974 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
 15975 specification” consistently.

15976 **NAME**

15977 gai_strerror — address and name information error description

15978 **SYNOPSIS**

15979 #include <netdb.h>

15980 const char *gai_strerror(int *ecode*);15981 **DESCRIPTION**15982 The *gai_strerror()* function shall return a text string describing an error value for the *getaddrinfo()*
15983 and *getnameinfo()* functions listed in the <netdb.h> header.15984 When the *ecode* argument is one of the following values listed in the <netdb.h> header:

15985 [EAI_AGAIN]

15986 [EAI_BADFLAGS]

15987 [EAI_FAIL]

15988 [EAI_FAMILY]

15989 [EAI_MEMORY]

15990 [EAI_NONAME]

15991 [EAI_OVERFLOW]

15992 [EAI_SERVICE]

15993 [EAI_SOCKTYPE]

15994 [EAI_SYSTEM]

15995 the function return value shall point to a string describing the error. If the argument is not one
15996 of those values, the function shall return a pointer to a string whose contents indicate an
15997 unknown error.15998 **RETURN VALUE**15999 Upon successful completion, *gai_strerror()* shall return a pointer to an implementation-defined
16000 string.16001 **ERRORS**

16002 No errors are defined.

16003 **EXAMPLES**

16004 None.

16005 **APPLICATION USAGE**

16006 None.

16007 **RATIONALE**

16008 None.

16009 **FUTURE DIRECTIONS**

16010 None.

16011 **SEE ALSO**16012 *getaddrinfo()*, the Base Definitions volume of IEEE Std 1003.1-2001, <netdb.h>16013 **CHANGE HISTORY**

16014 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

16015 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from
16016 **char *** to **const char ***. This is for coordination with the IPnG Working Group.16017 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the
16018 [EAI_OVERFLOW] error code. |

16019 **NAME**16020 gcvt — convert a floating-point number to a string (**LEGACY**)16021 **SYNOPSIS**

16022 xSI #include <stdlib.h>

16023 char *gcvt(double value, int ndigit, char *buf);

16024

16025 **DESCRIPTION**16026 Refer to *ecvt()*.

16027 **NAME**

16028 getaddrinfo — get address information

16029 **SYNOPSIS**

16030 #include <sys/socket.h>

16031 #include <netdb.h>

```
16032       int getaddrinfo(const char *restrict nodename,  
16033                      const char *restrict servname,  
16034                      const struct addrinfo *restrict hints,  
16035                      struct addrinfo **restrict res);
```

16036 **DESCRIPTION**16037 Refer to *freeaddrinfo()*.

16038 **NAME**

16039 getc — get a byte from a stream

16040 **SYNOPSIS**

16041 #include <stdio.h>

16042 int getc(FILE *stream);

16043 **DESCRIPTION**

16044 cx The functionality described on this reference page is aligned with the ISO C standard. Any
16045 conflict between the requirements described here and the ISO C standard is unintentional. This
16046 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16047 The *getc()* function shall be equivalent to *fgetc()*, except that if it is implemented as a macro it
16048 may evaluate *stream* more than once, so the argument should never be an expression with side
16049 effects.

16050 **RETURN VALUE**16051 Refer to *fgetc()*.16052 **ERRORS**16053 Refer to *fgetc()*.16054 **EXAMPLES**

16055 None.

16056 **APPLICATION USAGE**

16057 If the integer value returned by *getc()* is stored into a variable of type **char** and then compared
16058 against the integer constant EOF, the comparison may never succeed, because sign-extension of
16059 a variable of type **char** on widening to integer is implementation-defined.

16060 Since it may be implemented as a macro, *getc()* may treat incorrectly a *stream* argument with
16061 side effects. In particular, *getc(*f++)* does not necessarily work as expected. Therefore, use of this
16062 function should be preceded by "#undef getc" in such situations; *fgetc()* could also be used.

16063 **RATIONALE**

16064 None.

16065 **FUTURE DIRECTIONS**

16066 None.

16067 **SEE ALSO**16068 *fgetc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>16069 **CHANGE HISTORY**

16070 First released in Issue 1. Derived from Issue 1 of the SVID.

16071 **NAME**

16072 `getc_unlocked`, `getchar_unlocked`, `putc_unlocked`, `putchar_unlocked` — stdio with explicit client
 16073 locking

16074 **SYNOPSIS**

```
16075 TSF      #include <stdio.h>

16076      int getc_unlocked(FILE *stream);
16077      int getchar_unlocked(void);
16078      int putc_unlocked(int c, FILE *stream);
16079      int putchar_unlocked(int c);
16080
```

16081 **DESCRIPTION**

16082 Versions of the functions `getc()`, `getchar()`, `putc()`, and `putchar()` respectively named
 16083 `getc_unlocked()`, `getchar_unlocked()`, `putc_unlocked()`, and `putchar_unlocked()` shall be provided
 16084 which are functionally equivalent to the original versions, with the exception that they are not
 16085 required to be implemented in a thread-safe manner. They may only safely be used within a
 16086 scope protected by `flockfile()` (or `ftrylockfile()`) and `funlockfile()`. These functions may safely be
 16087 used in a multi-threaded program if and only if they are called while the invoking thread owns
 16088 the (**FILE ***) object, as is the case after a successful call to the `flockfile()` or `ftrylockfile()` functions.

16089 **RETURN VALUE**

16090 See `getc()`, `getchar()`, `putc()`, and `putchar()`.

16091 **ERRORS**

16092 See `getc()`, `getchar()`, `putc()`, and `putchar()`.

16093 **EXAMPLES**

16094 None.

16095 **APPLICATION USAGE**

16096 Since they may be implemented as macros, `getc_unlocked()` and `putc_unlocked()` may treat
 16097 incorrectly a *stream* argument with side effects. In particular, `getc_unlocked(*f++)` and
 16098 `putc_unlocked(*f++)` do not necessarily work as expected. Therefore, use of these functions in
 16099 such situations should be preceded by the following statement as appropriate:

```
16100      #undef getc_unlocked
16101      #undef putc_unlocked
```

16102 **RATIONALE**

16103 Some I/O functions are typically implemented as macros for performance reasons (for example,
 16104 `putc()` and `getc()`). For safety, they need to be synchronized, but it is often too expensive to
 16105 synchronize on every character. Nevertheless, it was felt that the safety concerns were more
 16106 important; consequently, the `getc()`, `getchar()`, `putc()`, and `putchar()` functions are required to be
 16107 thread-safe. However, unlocked versions are also provided with names that clearly indicate the
 16108 unsafe nature of their operation but can be used to exploit their higher performance. These
 16109 unlocked versions can be safely used only within explicitly locked program regions, using
 16110 exported locking primitives. In particular, a sequence such as:

```
16111      flockfile(fileptr);
16112      putc_unlocked('1', fileptr);
16113      putc_unlocked('\n', fileptr);
16114      fprintf(fileptr, "Line 2\n");
16115      funlockfile(fileptr);
```

16116 is permissible, and results in the text sequence:

16117 1
 16118 Line 2
 16119 being printed without being interspersed with output from other threads.

16120 It would be wrong to have the standard names such as *getc()*, *putc()*, and so on, map to the
 16121 “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still
 16122 want to inspect all uses of *getc()*, *putc()*, and so on, by hand when converting existing code.
 16123 Choosing the safe bindings as the default, at least, results in correct code and maintains the
 16124 “atomicity at the function” invariant. To do otherwise would introduce gratuitous
 16125 synchronization errors into converted code. Other routines that modify the *stdio* (**FILE ***)
 16126 structures or buffers are also safely synchronized.

16127 Note that there is no need for functions of the form *getc_locked()*, *putc_locked()*, and so on, since
 16128 this is the functionality of *getc()*, *putc()*, *et al.* It would be inappropriate to use a feature test
 16129 macro to switch a macro definition of *getc()* between *getc_locked()* and *getc_unlocked()*, since the
 16130 ISO C standard requires an actual function to exist, a function whose behavior could not be
 16131 changed by the feature test macro. Also, providing both the *xxx_locked()* and *xxx_unlocked()*
 16132 forms leads to the confusion of whether the suffix describes the behavior of the function or the
 16133 circumstances under which it should be used.

16134 Three additional routines, *flockfile()*, *ftrylockfile()*, and *funlockfile()* (which may be macros), are
 16135 provided to allow the user to delineate a sequence of I/O statements that are executed
 16136 synchronously.

16137 The *ungetc()* function is infrequently called relative to the other functions/macros so no
 16138 unlocked variation is needed.

16139 **FUTURE DIRECTIONS**
 16140 None.

16141 **SEE ALSO**
 16142 *getc()*, *getchar()*, *putc()*, *putchar()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 16143 **<stdio.h>**

16144 **CHANGE HISTORY**
 16145 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

16146 **Issue 6**
 16147 These functions are marked as part of the Thread-Safe Functions option.

16148 The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing
 16149 how applications should be written to avoid the case when the functions are implemented as
 16150 macros.

16151 **NAME**16152 `getchar` — get a byte from a stdin stream16153 **SYNOPSIS**16154 `#include <stdio.h>`16155 `int getchar(void);`16156 **DESCRIPTION**

16157 cx The functionality described on this reference page is aligned with the ISO C standard. Any
16158 conflict between the requirements described here and the ISO C standard is unintentional. This
16159 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16160 The `getchar()` function shall be equivalent to `getc(stdin)`.16161 **RETURN VALUE**16162 Refer to `fgetc()`.16163 **ERRORS**16164 Refer to `fgetc()`.16165 **EXAMPLES**

16166 None.

16167 **APPLICATION USAGE**

16168 If the integer value returned by `getchar()` is stored into a variable of type **char** and then
16169 compared against the integer constant EOF, the comparison may never succeed, because sign-
16170 extension of a variable of type **char** on widening to integer is implementation-defined.

16171 **RATIONALE**

16172 None.

16173 **FUTURE DIRECTIONS**

16174 None.

16175 **SEE ALSO**16176 `getc()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**16177 **CHANGE HISTORY**

16178 First released in Issue 1. Derived from Issue 1 of the SVID.

16179 **NAME**

16180 getchar_unlocked — stdio with explicit client locking

16181 **SYNOPSIS**

16182 TSF #include <stdio.h>

16183 int getchar_unlocked(void);

16184

16185 **DESCRIPTION**

16186 Refer to *getc_unlocked()*.

16187 **NAME**

16188 getcontext, setcontext — get and set current user context

16189 **SYNOPSIS**

16190 xSI #include <ucontext.h>

16191 int getcontext(ucontext_t *ucp);

16192 int setcontext(const ucontext_t *ucp);

16193

16194 **DESCRIPTION**

16195 The *getcontext()* function shall initialize the structure pointed to by *ucp* to the current user
 16196 context of the calling thread. The **ucontext_t** type that *ucp* points to defines the user context and
 16197 includes the contents of the calling thread's machine registers, the signal mask, and the current
 16198 execution stack.

16199 The *setcontext()* function shall restore the user context pointed to by *ucp*. A successful call to
 16200 *setcontext()* shall not return; program execution resumes at the point specified by the *ucp*
 16201 argument passed to *setcontext()*. The *ucp* argument should be created either by a prior call to
 16202 *getcontext()* or *makecontext()*, or by being passed as an argument to a signal handler. If the *ucp*
 16203 argument was created with *getcontext()*, program execution continues as if the corresponding
 16204 call of *getcontext()* had just returned. If the *ucp* argument was created with *makecontext()*,
 16205 program execution continues with the function passed to *makecontext()*. When that function
 16206 returns, the thread shall continue as if after a call to *setcontext()* with the *ucp* argument that was
 16207 input to *makecontext()*. If the *uc_link* member of the **ucontext_t** structure pointed to by the *ucp*
 16208 argument is equal to 0, then this context is the main context, and the thread shall exit when this
 16209 context returns. The effects of passing a *ucp* argument obtained from any other source are
 16210 unspecified.

16211 **RETURN VALUE**

16212 Upon successful completion, *setcontext()* shall not return and *getcontext()* shall return 0;
 16213 otherwise, a value of -1 shall be returned.

16214 **ERRORS**

16215 No errors are defined.

16216 **EXAMPLES**16217 Refer to *makecontext()*.16218 **APPLICATION USAGE**

16219 When a signal handler is executed, the current user context is saved and a new context is
 16220 created. If the thread leaves the signal handler via *longjmp()*, then it is unspecified whether the
 16221 context at the time of the corresponding *setjmp()* call is restored and thus whether future calls to
 16222 *getcontext()* provide an accurate representation of the current context, since the context restored
 16223 by *longjmp()* does not necessarily contain all the information that *setcontext()* requires. Signal
 16224 handlers should use *siglongjmp()* or *setcontext()* instead.

16225 Conforming applications should not modify or access the *uc_mcontext* member of **ucontext_t**. A
 16226 conforming application cannot assume that context includes any process-wide static data,
 16227 possibly including *errno*. Users manipulating contexts should take care to handle these
 16228 explicitly when required.

16229 Use of contexts to create alternate stacks is not defined by this volume of IEEE Std 1003.1-2001.

16230 **RATIONALE**

16231 None.

16232 **FUTURE DIRECTIONS**

16233 None.

16234 **SEE ALSO**16235 *bsd_signal()*, *makecontext()*, *setcontext()*, *setjmp()*, *sigaction()*, *sigaltstack()*, *siglongjmp()*,16236 *sigprocmask()*, *sigsetjmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<ucontext.h>**16237 **CHANGE HISTORY**

16238 First released in Issue 4, Version 2.

16239 **Issue 5**

16240 Moved from X/OPEN UNIX extension to BASE.

16241 The following sentence was removed from the DESCRIPTION: “If the *ucp* argument was passed
16242 to a signal handler, program execution continues with the program instruction following the
16243 instruction interrupted by the signal.”

16244 **NAME**

16245 getcwd — get the pathname of the current working directory

16246 **SYNOPSIS**

16247 #include <unistd.h>

16248 char *getcwd(char *buf, size_t size);

16249 **DESCRIPTION**

16250 The *getcwd()* function shall place an absolute pathname of the current working directory in the
 16251 array pointed to by *buf*, and return *buf*. The pathname copied to the array shall contain no
 16252 components that are symbolic links. The *size* argument is the size in bytes of the character array
 16253 pointed to by the *buf* argument. If *buf* is a null pointer, the behavior of *getcwd()* is unspecified.

16254 **RETURN VALUE**

16255 Upon successful completion, *getcwd()* shall return the *buf* argument. Otherwise, *getcwd()* shall
 16256 return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by
 16257 *buf* are then undefined.

16258 **ERRORS**16259 The *getcwd()* function shall fail if:16260 [EINVAL] The *size* argument is 0.

16261 [ERANGE] The *size* argument is greater than 0, but is smaller than the length of the
 16262 pathname +1.

16263 The *getcwd()* function may fail if:

16264 [EACCES] Read or search permission was denied for a component of the pathname.

16265 [ENOMEM] Insufficient storage space is available.

16266 **EXAMPLES**16267 **Determining the Absolute Pathname of the Current Working Directory**

16268 The following example returns a pointer to an array that holds the absolute pathname of the
 16269 current working directory. The pointer is returned in the *ptr* variable, which points to the *buf*
 16270 array where the pathname is stored.

16271 #include <stdlib.h>

16272 #include <unistd.h>

16273 ...

16274 long size;

16275 char *buf;

16276 char *ptr;

16277 size = pathconf(".", _PC_PATH_MAX);

16278 if ((buf = (char *)malloc((size_t)size)) != NULL)

16279 ptr = getcwd(buf, (size_t)size);

16280 ...

16281 **APPLICATION USAGE**

16282 None.

16283 RATIONALE

16284 Since the maximum pathname length is arbitrary unless {PATH_MAX} is defined, an application
16285 generally cannot supply a *buf* with size {{PATH_MAX}+1}.

16286 Having *getcwd()* take no arguments and instead use the *malloc()* function to produce space for
16287 the returned argument was considered. The advantage is that *getcwd()* knows how big the
16288 working directory pathname is and can allocate an appropriate amount of space. But the
16289 programmer would have to use the *free()* function to free the resulting object, or each use of
16290 *getcwd()* would further reduce the available memory. Also, *malloc()* and *free()* are used nowhere
16291 else in this volume of IEEE Std 1003.1-2001. Finally, *getcwd()* is taken from the SVID where it has
16292 the two arguments used in this volume of IEEE Std 1003.1-2001.

16293 The older function *getwd()* was rejected for use in this context because it had only a buffer
16294 argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except
16295 to depend on the programmer to provide a large enough buffer.

16296 On some implementations, if *buf* is a null pointer, *getcwd()* may obtain *size* bytes of memory
16297 using *malloc()*. In this case, the pointer returned by *getcwd()* may be used as the argument in a
16298 subsequent call to *free()*. Invoking *getcwd()* with *buf* as a null pointer is not recommended in
16299 conforming applications.

16300 If a program is operating in a directory where some (grand)parent directory does not permit
16301 reading, *getcwd()* may fail, as in most implementations it must read the directory to determine
16302 the name of the file. This can occur if search, but not read, permission is granted in an
16303 intermediate directory, or if the program is placed in that directory by some more privileged
16304 process (for example, login). Including the [EACCES] error condition makes the reporting of the
16305 error consistent and warns the application writer that *getcwd()* can fail for reasons beyond the
16306 control of the application writer or user. Some implementations can avoid this occurrence (for
16307 example, by implementing *getcwd()* using *pwd*, where *pwd* is a set-user-root process), thus the
16308 error was made optional. Since this volume of IEEE Std 1003.1-2001 permits the addition of other
16309 errors, this would be a common addition and yet one that applications could not be expected to
16310 deal with without this addition.

16311 FUTURE DIRECTIONS

16312 None.

16313 SEE ALSO

16314 *malloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

16315 CHANGE HISTORY

16316 First released in Issue 1. Derived from Issue 1 of the SVID.

16317 Issue 6

16318 The following new requirements on POSIX implementations derive from alignment with the
16319 Single UNIX Specification:

- 16320 • The [ENOMEM] optional error condition is added.

16321 **NAME**

16322 getdate — convert user format date and time

16323 **SYNOPSIS**16324 xSI `#include <time.h>`16325 `struct tm *getdate(const char *string);`

16326

16327 **DESCRIPTION**16328 The *getdate()* function shall convert a string representation of a date or time into a broken-down
16329 time.16330 The external variable or macro *getdate_err* is used by *getdate()* to return error values.16331 Templates are used to parse and interpret the input string. The templates are contained in a text
16332 file identified by the environment variable *DATMSK*. The *DATMSK* variable should be set to
16333 indicate the full pathname of the file that contains the templates. The first line in the template
16334 that matches the input specification is used for interpretation and conversion into the internal
16335 time format.

16336 The following conversion specifications shall be supported:

16337 %% Equivalent to %.

16338 %a Abbreviated weekday name.

16339 %A Full weekday name.

16340 %b Abbreviated month name.

16341 %B Full month name.

16342 %c Locale's appropriate date and time representation.

16343 %C Century number [00,99]; leading zeros are permitted but not required.

16344 %d Day of month [01,31]; the leading 0 is optional.

16345 %D Date as %m/%d/%y.

16346 %e Equivalent to %d.

16347 %h Abbreviated month name.

16348 %H Hour [00,23].

16349 %I Hour [01,12].

16350 %m Month number [01,12].

16351 %M Minute [00,59].

16352 %n Equivalent to <newline>.

16353 %p Locale's equivalent of either AM or PM.

16354 %r The locale's appropriate representation of time in AM and PM notation. In the POSIX
16355 locale, this shall be equivalent to %I:%M:%S %p.

16356 %R Time as %H:%M.

16357 %S Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap
16358 seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap
16359 second data must come from some external source.

16360	%t	Equivalent to <tab>.
16361	%T	Time as %H:%M:%S.
16362	%w	Weekday number (Sunday = [0,6]).
16363	%x	Locale's appropriate date representation.
16364	%X	Locale's appropriate time representation.
16365	%y	Year within century. When a century is not otherwise specified, values in the range
16366		[69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall
16367		refer to years 2000 to 2068 inclusive.
16368	Note:	It is expected that in a future version of IEEE Std 1003.1-2001 the default century
16369		inferred from a 2-digit year will change. (This would apply to all commands
16370		accepting a 2-digit year as input.)
16371	%Y	Year as "ccyy" (for example, 2001).
16372	%Z	Timezone name or no characters if no timezone exists. If the timezone supplied by %Z is
16373		not the timezone that <i>getdate()</i> expects, an invalid input specification error shall result.
16374		The <i>getdate()</i> function calculates an expected timezone based on information supplied
16375		to the function (such as the hour, day, and month).
16376	The match between the template and input specification performed by <i>getdate()</i> shall be case-	
16377	insensitive.	
16378	The month and weekday names can consist of any combination of upper and lowercase letters.	
16379	The process can request that the input date or time specification be in a specific language by	
16380	setting the <i>LC_TIME</i> category (see <i>setlocale()</i>).	
16381	Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most	
16382	two digits are allowed for those descriptors, including leading zeros. Extra whitespace in either	
16383	the template file or in <i>string</i> shall be ignored.	
16384	The results are undefined if the conversion specifications %c, %x, and %X include unsupported	
16385	conversion specifications.	
16386	The following rules apply for converting the input specification into the internal format:	
16387	• If %Z is being scanned, then <i>getdate()</i> shall initialize the broken-down time to be the current	
16388	time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on	
16389	the current local time as if <i>localtime()</i> had been called.	
16390	• If only the weekday is given, the day chosen shall be the day, starting with today and moving	
16391	into the future, which first matches the named day.	
16392	• If only the month (and no year) is given, the month chosen shall be the month, starting with	
16393	the current month and moving into the future, which first matches the named month. The	
16394	first day of the month shall be assumed if no day is given.	
16395	• If no hour, minute, and second are given, the current hour, minute, and second shall be	
16396	assumed.	
16397	• If no date is given, the hour chosen shall be the hour, starting with the current hour and	
16398	moving into the future, which first matches the named hour.	
16399	If a conversion specification in the <i>DATMSK</i> file does not correspond to one of the conversion	
16400	specifications above, the behavior is unspecified.	
16401	The <i>getdate()</i> function need not be reentrant. A function that is not required to be reentrant is not	
16402	required to be thread-safe.	

16403 **RETURN VALUE**

16404 Upon successful completion, *getdate()* shall return a pointer to a **struct tm**. Otherwise, it shall
 16405 return a null pointer and set *getdate_err* to indicate the error.

16406 **ERRORS**

16407 The *getdate()* function shall fail in the following cases, setting *getdate_err* to the value shown in
 16408 the list below. Any changes to *errno* are unspecified.

- 16409 1. The *DATETIME* environment variable is null or undefined.
- 16410 2. The template file cannot be opened for reading.
- 16411 3. Failed to get file status information.
- 16412 4. The template file is not a regular file.
- 16413 5. An I/O error is encountered while reading the template file.
- 16414 6. Memory allocation failed (not enough memory available).
- 16415 7. There is no line in the template that matches the input.
- 16416 8. Invalid input specification. For example, February 31; or a time is specified that cannot be
 16417 represented in a **time_t** (representing the time in seconds since the Epoch).

16418 **EXAMPLES**

- 16419 1. The following example shows the possible contents of a template:

```
16420       %m
16421       %A %B %d, %Y, %H:%M:%S
16422       %A
16423       %B
16424       %m/%d/%Y %I %p
16425       %d,%m,%Y %H:%M
16426       at %A the %dst of %B in %Y
16427       run job at %I %p,%B %dnd
16428       %A den %d. %B %Y %H.%M Uhr
```

- 16429 2. The following are examples of valid input specifications for the template in Example 1:

```
16430       getdate("10/1/87 4 PM");
16431       getdate("Friday");
16432       getdate("Friday September 18, 1987, 10:30:30");
16433       getdate("24,9,1986 10:30");
16434       getdate("at monday the 1st of december in 1986");
16435       getdate("run job at 3 PM, december 2nd");
```

16436 If the *LC_TIME* category is set to a German locale that includes *freitag* as a weekday name
 16437 and *oktober* as a month name, the following would be valid:

```
16438       getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

- 16439 3. The following example shows how local date and time specification can be defined in the
 16440 template:

16441
16442
16443
16444
16445
16446

Invocation	Line in Template
getdate("11/27/86")	%m/%d/%y
getdate("27.11.86")	%d.%m.%y
getdate("86-11-27")	%y-%m-%d
getdate("Friday 12:00:00")	%A %H:%M:%S

16447
16448
16449
16450

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC_TIME* category is set to the default C locale:

16451
16452
16453
16454
16455
16456
16457
16458
16459
16460
16461
16462
16463
16464

Input	Line in Template	Date
Mon	%a	Mon Sep 22 12:19:47 EDT 1986
Sun	%a	Sun Sep 28 12:19:47 EDT 1986
Fri	%a	Fri Sep 26 12:19:47 EDT 1986
September	%B	Mon Sep 1 12:19:47 EDT 1986
January	%B	Thu Jan 1 12:19:47 EST 1987
December	%B	Mon Dec 1 12:19:47 EST 1986
Sep Mon	%b %a	Mon Sep 1 12:19:47 EDT 1986
Jan Fri	%b %a	Fri Jan 2 12:19:47 EST 1987
Dec Mon	%b %a	Mon Dec 1 12:19:47 EST 1986
Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:47 EST 1989
Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

16465 APPLICATION USAGE

16466 Although historical versions of *getdate()* did not require that **<time.h>** declare the external
16467 variable *getdate_err*, this volume of IEEE Std 1003.1-2001 does require it. The standard
16468 developers encourage applications to remove declarations of *getdate_err* and instead incorporate
16469 the declaration by including **<time.h>**.

16470 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

16471 RATIONALE

16472 In standard locales, the conversion specifications %c, %x, and %X do not include unsupported
16473 conversion specifiers and so the text regarding results being undefined is not a problem in that
16474 case.

16475 FUTURE DIRECTIONS

16476 None.

16477 SEE ALSO

16478 *ctime()*, *localtime()*, *setlocale()*, *strftime()*, *times()*, the Base Definitions volume of
16479 IEEE Std 1003.1-2001, **<time.h>**

16480 CHANGE HISTORY

16481 First released in Issue 4, Version 2.

16482 Issue 5

16483 Moved from X/OPEN UNIX extension to BASE.

16484 The last paragraph of the DESCRIPTION is added.

16485 The %C conversion specification is added, and the exact meaning of the %y conversion
16486 specification is clarified in the DESCRIPTION.

- 16487 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 16488 The %R conversion specification is changed to follow historical practice.
- 16489 **Issue 6**
- 16490 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since
- 16491 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*
- 16492 functions.
- 16493 The description of %S is updated so that the valid range is [00,60] rather than [00,61].
- 16494 The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors
- 16495 for consistency with other functions.

16496 **NAME**

16497 getegid — get the effective group ID

16498 **SYNOPSIS**

16499 #include <unistd.h>

16500 gid_t getegid(void);

16501 **DESCRIPTION**

16502 The *getegid()* function shall return the effective group ID of the calling process.

16503 **RETURN VALUE**

16504 The *getegid()* function shall always be successful and no return value is reserved to indicate an
16505 error.

16506 **ERRORS**

16507 No errors are defined.

16508 **EXAMPLES**

16509 None.

16510 **APPLICATION USAGE**

16511 None.

16512 **RATIONALE**

16513 None.

16514 **FUTURE DIRECTIONS**

16515 None.

16516 **SEE ALSO**

16517 *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base
16518 Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

16519 **CHANGE HISTORY**

16520 First released in Issue 1. Derived from Issue 1 of the SVID.

16521 **Issue 6**

16522 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16523 The following new requirements on POSIX implementations derive from alignment with the
16524 Single UNIX Specification:

- 16525
 - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
16526 required for conforming implementations of previous POSIX specifications, it was not
16527 required for UNIX applications.

16528 **NAME**

16529 getenv — get value of an environment variable

16530 **SYNOPSIS**

16531 #include <stdlib.h>

16532 char *getenv(const char *name);

16533 **DESCRIPTION**

16534 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 16535 conflict between the requirements described here and the ISO C standard is unintentional. This
 16536 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16537 The *getenv()* function shall search the environment of the calling process (see the Base
 16538 Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables) for the
 16539 environment variable *name* if it exists and return a pointer to the value of the environment
 16540 variable. If the specified environment variable cannot be found, a null pointer shall be returned.
 16541 The application shall ensure that it does not modify the string pointed to by the *getenv()*
 16542 function.

16543 CX The string pointed to may be overwritten by a subsequent call to *getenv()*, *setenv()*, or *unsetenv()*,
 16544 but shall not be overwritten by a call to any other function in this volume of
 16545 IEEE Std 1003.1-2001.

16546 CX If the application modifies *environ* or the pointers to which it points, the behavior of *getenv()* is
 16547 undefined.

16548 The *getenv()* function need not be reentrant. A function that is not required to be reentrant is not
 16549 required to be thread-safe.

16550 **RETURN VALUE**

16551 Upon successful completion, *getenv()* shall return a pointer to a string containing the *value* for
 16552 the specified *name*. If the specified *name* cannot be found in the environment of the calling
 16553 process, a null pointer shall be returned.

16554 The return value from *getenv()* may point to static data which may be overwritten by
 16555 CX subsequent calls to *getenv()*, *setenv()*, or *unsetenv()*.

16556 XSI On XSI-conformant systems, the return value from *getenv()* may point to static data which may
 16557 also be overwritten by subsequent calls to *putenv()*.

16558 **ERRORS**

16559 No errors are defined.

16560 **EXAMPLES**16561 **Getting the Value of an Environment Variable**16562 The following example gets the value of the *HOME* environment variable.

16563 #include <stdlib.h>

16564 ...

16565 const char *name = "HOME";

16566 char *value;

16567 value = getenv(name);

16568 **APPLICATION USAGE**

16569 None.

16570 **RATIONALE**

16571 The *clearenv()* function was considered but rejected. The *putenv()* function has now been
16572 included for alignment with the Single UNIX Specification.

16573 The *getenv()* function is inherently not reentrant because it returns a value pointing to static
16574 data.

16575 Conforming applications are required not to modify *environ* directly, but to use only the
16576 functions described here to manipulate the process environment as an abstract object. Thus, the
16577 implementation of the environment access functions has complete control over the data
16578 structure used to represent the environment (subject to the requirement that *environ* be
16579 maintained as a list of strings with embedded equal signs for applications that wish to scan the
16580 environment). This constraint allows the implementation to properly manage the memory it
16581 allocates, either by using allocated storage for all variables (copying them on the first invocation
16582 of *setenv()* or *unsetenv()*), or keeping track of which strings are currently in allocated space and
16583 which are not, via a separate table or some other means. This enables the implementation to free
16584 any allocated space used by strings (and perhaps the pointers to them) stored in *environ* when
16585 *unsetenv()* is called. A C runtime start-up procedure (that which invokes *main()* and perhaps
16586 initializes *environ*) can also initialize a flag indicating that none of the environment has yet been
16587 copied to allocated storage, or that the separate table has not yet been initialized.

16588 In fact, for higher performance of *getenv()*, the implementation could also maintain a separate
16589 copy of the environment in a data structure that could be searched much more quickly (such as
16590 an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when
16591 *setenv()* or *unsetenv()* is invoked.

16592 Performance of *getenv()* can be important for applications which have large numbers of
16593 environment variables. Typically, applications like this use the environment as a resource
16594 database of user-configurable parameters. The fact that these variables are in the user's shell
16595 environment usually means that any other program that uses environment variables (such as *ls*,
16596 which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC_ALL*, and so on) is
16597 similarly slowed down by the linear search through the variables.

16598 An implementation that maintains separate data structures, or even one that manages the
16599 memory it consumes, is not currently required as it was thought it would reduce consensus
16600 among implementors who do not want to change their historical implementations.

16601 The POSIX Threads Extension states that multi-threaded applications must not modify *environ*
16602 directly, and that IEEE Std 1003.1-2001 is providing functions which such applications can use in
16603 the future to manipulate the environment in a thread-safe manner. Thus, moving away from
16604 application use of *environ* is desirable from that standpoint as well.

16605 **FUTURE DIRECTIONS**

16606 None.

16607 **SEE ALSO**

16608 *exec*, *putenv()*, *setenv()*, *unsetenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter
16609 8, Environment Variables, <**stdlib.h**>

16610 **CHANGE HISTORY**

16611 First released in Issue 1. Derived from Issue 1 of the SVID.

16612 Issue 5

16613 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
16614 VALUE section.

16615 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

16616 Issue 6

16617 The following changes were made to align with the IEEE P1003.1a draft standard:

- 16618 • References added to the new *setenv()* and *unsetenv()* functions.

16619 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

16620 **NAME**

16621 geteuid — get the effective user ID

16622 **SYNOPSIS**

16623 #include <unistd.h>

16624 uid_t geteuid(void);

16625 **DESCRIPTION**

16626 The *geteuid()* function shall return the effective user ID of the calling process.

16627 **RETURN VALUE**

16628 The *geteuid()* function shall always be successful and no return value is reserved to indicate an
16629 error.

16630 **ERRORS**

16631 No errors are defined.

16632 **EXAMPLES**

16633 None.

16634 **APPLICATION USAGE**

16635 None.

16636 **RATIONALE**

16637 None.

16638 **FUTURE DIRECTIONS**

16639 None.

16640 **SEE ALSO**

16641 *getegid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base
16642 Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

16643 **CHANGE HISTORY**

16644 First released in Issue 1. Derived from Issue 1 of the SVID.

16645 **Issue 6**

16646 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16647 The following new requirements on POSIX implementations derive from alignment with the
16648 Single UNIX Specification:

- 16649
 - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
16650 required for conforming implementations of previous POSIX specifications, it was not
16651 required for UNIX applications.

16652 **NAME**

16653 getgid — get the real group ID

16654 **SYNOPSIS**

16655 #include <unistd.h>

16656 gid_t getgid(void);

16657 **DESCRIPTION**16658 The *getgid()* function shall return the real group ID of the calling process.16659 **RETURN VALUE**16660 The *getgid()* function shall always be successful and no return value is reserved to indicate an
16661 error.16662 **ERRORS**

16663 No errors are defined.

16664 **EXAMPLES**

16665 None.

16666 **APPLICATION USAGE**

16667 None.

16668 **RATIONALE**

16669 None.

16670 **FUTURE DIRECTIONS**

16671 None.

16672 **SEE ALSO**16673 *getegid()*, *geteuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base
16674 Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>16675 **CHANGE HISTORY**

16676 First released in Issue 1. Derived from Issue 1 of the SVID.

16677 **Issue 6**

16678 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16679 The following new requirements on POSIX implementations derive from alignment with the
16680 Single UNIX Specification:

- 16681
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
16682 required for conforming implementations of previous POSIX specifications, it was not
16683 required for UNIX applications.

16684 **NAME**

16685 getgrent — get the group database entry

16686 **SYNOPSIS**

16687 XSI #include <grp.h>

16688 struct group *getgrent(void);

16689

16690 **DESCRIPTION**

16691 Refer to *endgrent()*.

16692 **NAME**

16693 getgrgid, getgrgid_r — get group database entry for a group ID

16694 **SYNOPSIS**

16695 #include <grp.h>

16696 struct group *getgrgid(gid_t gid);

16697 TSF int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
16698 size_t bufsize, struct group **result);

16699

16700 **DESCRIPTION**16701 The *getgrgid()* function shall search the group database for an entry with a matching *gid*.16702 The *getgrgid()* function need not be reentrant. A function that is not required to be reentrant is
16703 not required to be thread-safe.16704 TSF The *getgrgid_r()* function shall update the **group** structure pointed to by *grp* and store a pointer
16705 to that structure at the location pointed to by *result*. The structure shall contain an entry from
16706 the group database with a matching *gid*. Storage referenced by the group structure is allocated
16707 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The
16708 maximum size needed for this buffer can be determined with the {_SC_GETGR_R_SIZE_MAX}
16709 sysconf() parameter. A NULL pointer shall be returned at the location pointed to by *result* on
16710 error or if the requested entry is not found.16711 **RETURN VALUE**16712 Upon successful completion, *getgrgid()* shall return a pointer to a **struct group** with the structure
16713 defined in <grp.h> with a matching entry if one is found. The *getgrgid()* function shall return a
16714 null pointer if either the requested entry was not found, or an error occurred. On error, *errno*
16715 shall be set to indicate the error.16716 The return value may point to a static area which is overwritten by a subsequent call to
16717 *getgrent()*, *getgrgid()*, or *getgrnam()*.16718 TSF If successful, the *getgrgid_r()* function shall return zero; otherwise, an error number shall be
16719 returned to indicate the error.16720 **ERRORS**16721 The *getgrgid()* and *getgrgid_r()* functions may fail if:

16722 [EIO] An I/O error has occurred.

16723 [EINTR] A signal was caught during *getgrgid()*.

16724 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

16725 [ENFILE] The maximum allowable number of files is currently open in the system.

16726 TSF The *getgrgid_r()* function may fail if:16727 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to
16728 be referenced by the resulting **group** structure.

16729 **EXAMPLES**16730 **Finding an Entry in the Group Database**

16731 The following example uses *getgrgid()* to search the group database for a group ID that was
 16732 previously stored in a **stat** structure, then prints out the group name if it is found. If the group is
 16733 not found, the program prints the numeric value of the group for the entry.

```
16734 #include <sys/types.h>
16735 #include <grp.h>
16736 #include <stdio.h>
16737 ...
16738 struct stat statbuf;
16739 struct group *grp;
16740 ...
16741 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
16742     printf(" %-8.8s", grp->gr_name);
16743 else
16744     printf(" %-8d", statbuf.st_gid);
16745 ...
```

16746 **APPLICATION USAGE**

16747 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid()*.
 16748 If *errno* is set on return, an error occurred.

16749 The *getgrgid_r()* function is thread-safe and shall return values in a user-supplied buffer instead
 16750 of possibly using a static data area that may be overwritten by each call.

16751 **RATIONALE**

16752 None.

16753 **FUTURE DIRECTIONS**

16754 None.

16755 **SEE ALSO**

16756 *endgrent()*, *getgrnam()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<grp.h>**,
 16757 **<limits.h>**, **<sys/types.h>**

16758 **CHANGE HISTORY**

16759 First released in Issue 1. Derived from System V Release 2.0.

16760 **Issue 5**

16761 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 16762 VALUE section.

16763 The *getgrgid_r()* function is included for alignment with the POSIX Threads Extension.

16764 A note indicating that the *getgrgid()* function need not be reentrant is added to the
 16765 DESCRIPTION.

16766 **Issue 6**

16767 The *getgrgid_r()* function is marked as part of the Thread-Safe Functions option.

16768 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION
 16769 describing matching the *gid*.

16770 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

16771 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

- 16772 The following new requirements on POSIX implementations derive from alignment with the
16773 Single UNIX Specification:
- 16774 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
16775 required for conforming implementations of previous POSIX specifications, it was not
16776 required for UNIX applications.
 - 16777 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
 - 16778 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.
- 16779 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
16780 its avoidance of possibly using a static data area.
- 16781 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
16782 buffer from *bufsize* characters to bytes.

16783 **NAME**

16784 getgrnam, getgrnam_r — search group database for a name

16785 **SYNOPSIS**

16786 #include <grp.h>

16787 struct group *getgrnam(const char *name);

16788 TSF int getgrnam_r(const char *name, struct group *grp, char *buffer,
16789 size_t bufsize, struct group **result);

16790

16791 **DESCRIPTION**16792 The *getgrnam()* function shall search the group database for an entry with a matching *name*.16793 The *getgrnam()* function need not be reentrant. A function that is not required to be reentrant is
16794 not required to be thread-safe.16795 TSF The *getgrnam_r()* function shall update the **group** structure pointed to by *grp* and store a pointer
16796 to that structure at the location pointed to by *result*. The structure shall contain an entry from
16797 the group database with a matching *gid* or *name*. Storage referenced by the **group** structure is
16798 allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The
16799 maximum size needed for this buffer can be determined with the {_SC_GETGR_R_SIZE_MAX}
16800 sysconf() parameter. A NULL pointer is returned at the location pointed to by *result* on error or if
16801 the requested entry is not found.16802 **RETURN VALUE**16803 The *getgrnam()* function shall return a pointer to a **struct group** with the structure defined in
16804 <grp.h> with a matching entry if one is found. The *getgrnam()* function shall return a null
16805 pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be
16806 set to indicate the error.16807 The return value may point to a static area which is overwritten by a subsequent call to
16808 *getgrent()*, *getgrgid()*, or *getgrnam()*.16809 TSF If successful, the *getgrnam_r()* function shall return zero; otherwise, an error number shall be
16810 returned to indicate the error.16811 **ERRORS**16812 The *getgrnam()* and *getgrnam_r()* functions may fail if:

16813 [EIO] An I/O error has occurred.

16814 [EINTR] A signal was caught during *getgrnam()*.

16815 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

16816 [ENFILE] The maximum allowable number of files is currently open in the system.

16817 The *getgrnam_r()* function may fail if:16818 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to
16819 be referenced by the resulting **group** structure.

16820 **EXAMPLES**

16821 None.

16822 **APPLICATION USAGE**

16823 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam()*.
16824 If *errno* is set on return, an error occurred.

16825 The *getgrnam_r()* function is thread-safe and shall return values in a user-supplied buffer instead
16826 of possibly using a static data area that may be overwritten by each call.

16827 **RATIONALE**

16828 None.

16829 **FUTURE DIRECTIONS**

16830 None.

16831 **SEE ALSO**

16832 *endgrent()*, *getgrgid()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<grp.h>**, **<limits.h>**,
16833 **<sys/types.h>**

16834 **CHANGE HISTORY**

16835 First released in Issue 1. Derived from System V Release 2.0.

16836 **Issue 5**

16837 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
16838 VALUE section.

16839 The *getgrnam_r()* function is included for alignment with the POSIX Threads Extension.

16840 A note indicating that the *getgrnam()* function need not be reentrant is added to the
16841 DESCRIPTION.

16842 **Issue 6**

16843 The *getgrnam_r()* function is marked as part of the Thread-Safe Functions option.

16844 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

16845 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

16846 The following new requirements on POSIX implementations derive from alignment with the
16847 Single UNIX Specification:

16848

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
16849 required for conforming implementations of previous POSIX specifications, it was not
16850 required for UNIX applications.

16851

- In the RETURN VALUE section, the requirement to set *errno* on error is added.

16852

- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

16853 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
16854 its avoidance of possibly using a static data area.

16855 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
16856 buffer from *bufsize* characters to bytes.

16857 **NAME**

16858 getgroups — get supplementary group IDs

16859 **SYNOPSIS**

16860 #include <unistd.h>

16861 int getgroups(int *gidsetsize*, gid_t *grouplist*[]);16862 **DESCRIPTION**

16863 The *getgroups()* function shall fill in the array *grouplist* with the current supplementary group
 16864 IDs of the calling process. It is implementation-defined whether *getgroups()* also returns the
 16865 effective group ID in the *grouplist* array.

16866 The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual
 16867 number of group IDs stored in the array shall be returned. The values of array entries with
 16868 indices greater than or equal to the value returned are undefined.

16869 If *gidsetsize* is 0, *getgroups()* shall return the number of group IDs that it would otherwise return
 16870 without modifying the array pointed to by *grouplist*.

16871 If the effective group ID of the process is returned with the supplementary group IDs, the value
 16872 returned shall always be greater than or equal to one and less than or equal to the value of
 16873 {NGROUPS_MAX}+1.

16874 **RETURN VALUE**

16875 Upon successful completion, the number of supplementary group IDs shall be returned. A
 16876 return value of -1 indicates failure and *errno* shall be set to indicate the error.

16877 **ERRORS**16878 The *getgroups()* function shall fail if:

16879 [EINVAL] The *gidsetsize* argument is non-zero and less than the number of group IDs
 16880 that would have been returned.

16881 **EXAMPLES**16882 **Getting the Supplementary Group IDs of the Calling Process**

16883 The following example places the current supplementary group IDs of the calling process into
 16884 the *group* array.

```
16885       #include <sys/types.h>
16886       #include <unistd.h>
16887       ...
16888       gid_t *group;
16889       int nogroups;
16890       long ngroups_max;

16891       ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
16892       group = (gid_t *)malloc(ngroups_max * sizeof(gid_t));
16893       ngroups = getgroups(ngroups_max, group);
```

16894 **APPLICATION USAGE**

16895 None.

16896 **RATIONALE**

16897 The related function *setgroups()* is a privileged operation and therefore is not covered by this
 16898 volume of IEEE Std 1003.1-2001.

As implied by the definition of supplementary groups, the effective group ID may appear in the array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but the application needs to call *getegid()* to be sure of getting all of the information. Various implementation variations and administrative sequences cause the set of groups appearing in the result of *getgroups()* to vary in order and as to whether the effective group ID is included, even when the set of groups is the same (in the mathematical sense of “set”). (The history of a process and its parents could affect the details of the result.)

Application writers should note that {NGROUPS_MAX} is not necessarily a constant on all implementations.

16908 **FUTURE DIRECTIONS**

16909 None.

16910 **SEE ALSO**

16911 *getegid()*, *setgid()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**,
16912 **<unistd.h>**

16913 **CHANGE HISTORY**

16914 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

16915 **Issue 5**

16916 Normative text previously in the APPLICATION USAGE section is moved to the
16917 DESCRIPTION.

16918 **Issue 6**

16919 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

16920 The following new requirements on POSIX implementations derive from alignment with the
16921 Single UNIX Specification:

- 16922 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
16923 required for conforming implementations of previous POSIX specifications, it was not
16924 required for UNIX applications.
- 16925 • A return value of 0 is not permitted, because {NGROUPS_MAX} cannot be 0. This is a FIPS
16926 requirement.

16927 The following changes were made to align with the IEEE P1003.1a draft standard:

- 16928 • An explanation is added that the effective group ID may be included in the supplementary
16929 group list.

16930 **NAME**

16931 gethostbyaddr, gethostbyname — network host database functions

16932 **SYNOPSIS**

16933 #include <netdb.h>

```
16934 OB struct hostent *gethostbyaddr(const void *addr, socklen_t len,
16935 int type);
16936 struct hostent *gethostbyname(const char *name);
16937
```

16938 **DESCRIPTION**

16939 These functions shall retrieve information about hosts. This information is considered to be
 16940 stored in a database that can be accessed sequentially or randomly. Implementation of this
 16941 database is unspecified.

16942 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,
 16943 RFC 1035, and RFC 1886.

16944 Entries shall be returned in **hostent** structures.

16945 The *gethostbyaddr()* function shall return an entry containing addresses of address family *type* for
 16946 the host with address *addr*. The *len* argument contains the length of the address pointed to by
 16947 *addr*. The *gethostbyaddr()* function need not be reentrant. A function that is not required to be
 16948 reentrant is not required to be thread-safe.

16949 The *gethostbyname()* function shall return an entry containing addresses of address family
 16950 AF_INET for the host with name *name*. The *gethostbyname()* function need not be reentrant. A
 16951 function that is not required to be reentrant is not required to be thread-safe.

16952 The *addr* argument of *gethostbyaddr()* shall be an **in_addr** structure when *type* is AF_INET. It
 16953 contains a binary format (that is, not null-terminated) address in network byte order. The
 16954 *gethostbyaddr()* function is not guaranteed to return addresses of address families other than
 16955 AF_INET, even when such addresses exist in the database.

16956 If *gethostbyaddr()* returns successfully, then the *h_addrtype* field in the result shall be the same as
 16957 the *type* argument that was passed to the function, and the *h_addr_list* field shall list a single
 16958 address that is a copy of the *addr* argument that was passed to the function.

16959 The *name* argument of *gethostbyname()* shall be a node name; the behavior of *gethostbyname()*
 16960 when passed a numeric address string is unspecified. For IPv4, a numeric address string shall be
 16961 in the dotted-decimal notation described in *inet_addr()*.

16962 If *name* is not a numeric address string and is an alias for a valid host name, then *gethostbyname()*
 16963 shall return information about the host name to which the alias refers, and *name* shall be
 16964 included in the list of aliases returned.

16965 **RETURN VALUE**

16966 Upon successful completion, these functions shall return a pointer to a **hostent** structure if the
 16967 requested entry was found, and a null pointer if the end of the database was reached or the
 16968 requested entry was not found.

16969 Upon unsuccessful completion, *gethostbyaddr()* and *gethostbyname()* shall set *h_errno* to indicate
 16970 the error.

16971 **ERRORS**

16972 These functions shall fail in the following cases. The *gethostbyaddr()* and *gethostbyname()*
 16973 functions shall set *h_errno* to the value shown in the list below. Any changes to *errno* are
 16974 unspecified.

16975 [HOST_NOT_FOUND]
 16976 No such host is known.

16977 [NO_DATA] The server recognized the request and the name, but no address is available.
 16978 Another type of request to the name server for the domain might return an
 16979 answer.

16980 [NO_RECOVERY]
 16981 An unexpected server failure occurred which cannot be recovered.

16982 [TRY_AGAIN] A temporary and possibly transient error occurred, such as a failure of a
 16983 server to respond.

16984 **EXAMPLES**

16985 None.

16986 **APPLICATION USAGE**

16987 The *gethostbyaddr()* and *gethostbyname()* functions may return pointers to static data, which may
 16988 be overwritten by subsequent calls to any of these functions.

16989 The *getaddrinfo()* and *getnameinfo()* functions are preferred over the *gethostbyaddr()* and
 16990 *gethostbyname()* functions.

16991 **RATIONALE**

16992 None.

16993 **FUTURE DIRECTIONS**

16994 The *gethostbyaddr()* and *gethostbyname()* functions may be withdrawn in a future version.

16995 **SEE ALSO**

16996 *endhostent()*, *endservent()*, *gai_strerror()*, *getaddrinfo()*, *h_errno*, *inet_addr()*, the Base Definitions
 16997 volume of IEEE Std 1003.1-2001, **<netdb.h>**

16998 **CHANGE HISTORY**

16999 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17000 **NAME**

17001 gethostent — network host database functions

17002 **SYNOPSIS**

17003 #include <netdb.h>

17004 struct hostent *gethostent(void);

17005 **DESCRIPTION**

17006 Refer to *endhostent()*.

17007 **NAME**

17008 gethostid — get an identifier for the current host

17009 **SYNOPSIS**

17010 xSI #include <unistd.h>

17011 long gethostid(void);

17012

17013 **DESCRIPTION**

17014 The *gethostid()* function shall retrieve a 32-bit identifier for the current host.

17015 **RETURN VALUE**

17016 Upon successful completion, *gethostid()* shall return an identifier for the current host.

17017 **ERRORS**

17018 No errors are defined.

17019 **EXAMPLES**

17020 None.

17021 **APPLICATION USAGE**

17022 This volume of IEEE Std 1003.1-2001 does not define the domain in which the return value is
17023 unique.

17024 **RATIONALE**

17025 None.

17026 **FUTURE DIRECTIONS**

17027 None.

17028 **SEE ALSO**

17029 *random()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

17030 **CHANGE HISTORY**

17031 First released in Issue 4, Version 2.

17032 **Issue 5**

17033 Moved from X/OPEN UNIX extension to BASE.

17034 **NAME**

17035 gethostname — get name of current host

17036 **SYNOPSIS**

17037 #include <unistd.h>

17038 int gethostname(char *name, size_t namelen);

17039 **DESCRIPTION**

17040 The *gethostname()* function shall return the standard host name for the current machine. The
17041 *namelen* argument shall specify the size of the array pointed to by the *name* argument. The
17042 returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold
17043 the host name, then the returned name shall be truncated and it is unspecified whether the
17044 returned name is null-terminated.

17045 Host names are limited to {HOST_NAME_MAX} bytes.

17046 **RETURN VALUE**

17047 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

17048 **ERRORS**

17049 No errors are defined.

17050 **EXAMPLES**

17051 None.

17052 **APPLICATION USAGE**

17053 None.

17054 **RATIONALE**

17055 None.

17056 **FUTURE DIRECTIONS**

17057 None.

17058 **SEE ALSO**17059 *gethostid()*, *uname()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>17060 **CHANGE HISTORY**

17061 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17062 The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from
17063 **socklen_t** to **size_t**.

17064 **NAME**

17065 getitimer, setitimer — get and set value of interval timer

17066 **SYNOPSIS**

17067 XSI #include <sys/time.h>

17068 int getitimer(int which, struct itimerval *value);

17069 int setitimer(int which, const struct itimerval *restrict value,
17070 struct itimerval *restrict ovalue);

17071

17072 **DESCRIPTION**

17073 The *getitimer()* function shall store the current value of the timer specified by *which* into the
 17074 structure pointed to by *value*. The *setitimer()* function shall set the timer specified by *which* to
 17075 the value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, store
 17076 the previous value of the timer in the structure pointed to by *ovalue*.

17077 A timer value is defined by the **itimerval** structure, specified in <sys/time.h>. If *it_value* is non-
 17078 zero, it shall indicate the time to the next timer expiration. If *it_interval* is non-zero, it shall
 17079 specify a value to be used in reloading *it_value* when the timer expires. Setting *it_value* to 0 shall
 17080 disable a timer, regardless of the value of *it_interval*. Setting *it_interval* to 0 shall disable a timer
 17081 after its next expiration (assuming *it_value* is non-zero).

17082 Implementations may place limitations on the granularity of timer values. For each interval
 17083 timer, if the requested timer value requires a finer granularity than the implementation supports,
 17084 the actual timer value shall be rounded up to the next supported value.

17085 An XSI-conforming implementation provides each process with at least three interval timers,
 17086 which are indicated by the *which* argument:

17087 ITIMER_REAL Decrements in real time. A SIGALRM signal is delivered when this timer
 17088 expires.

17089 ITIMER_VIRTUAL Decrements in process virtual time. It runs only when the process is
 17090 executing. A SIGVTALRM signal is delivered when it expires.

17091 ITIMER_PROF Decrements both in process virtual time and when the system is running
 17092 on behalf of the process. It is designed to be used by interpreters in
 17093 statistically profiling the execution of interpreted programs. Each time the
 17094 ITIMER_PROF timer expires, the SIGPROF signal is delivered.

17095 The interaction between *setitimer()* and any of *alarm()*, *sleep()*, or *usleep()* is unspecified.

17096 **RETURN VALUE**

17097 Upon successful completion, *getitimer()* or *setitimer()* shall return 0; otherwise, -1 shall be
 17098 returned and *errno* set to indicate the error.

17099 **ERRORS**

17100 The *setitimer()* function shall fail if:

17101 [EINVAL] The *value* argument is not in canonical form. (In canonical form, the number of
 17102 microseconds is a non-negative integer less than 1 000 000 and the number of
 17103 seconds is a non-negative integer.)

17104 The *getitimer()* and *setitimer()* functions may fail if:

17105 [EINVAL] The *which* argument is not recognized.

17106 **EXAMPLES**

17107 None.

17108 **APPLICATION USAGE**

17109 None.

17110 **RATIONALE**

17111 None.

17112 **FUTURE DIRECTIONS**

17113 None.

17114 **SEE ALSO**

17115 *alarm()*, *sleep()*, *timer_getoverrun()*, *ualarm()*, *usleep()*, the Base Definitions volume of
17116 IEEE Std 1003.1-2001, <**signal.h**>, <**sys/time.h**>

17117 **CHANGE HISTORY**

17118 First released in Issue 4, Version 2.

17119 **Issue 5**

17120 Moved from X/OPEN UNIX extension to BASE.

17121 **Issue 6**

17122 The **restrict** keyword is added to the *setitimer()* prototype for alignment with the
17123 ISO/IEC 9899:1999 standard.

17124 **NAME**

17125 getlogin, getlogin_r — get login name

17126 **SYNOPSIS**

17127 #include <unistd.h>

17128 char *getlogin(void);

17129 TSF int getlogin_r(char *name, size_t namesize);

17130

17131 **DESCRIPTION**

17132 The *getlogin()* function shall return a pointer to a string containing the user name associated by
 17133 the login activity with the controlling terminal of the current process. If *getlogin()* returns a non-
 17134 null pointer, then that pointer points to the name that the user logged in under, even if there are
 17135 several login names with the same user ID.

17136 The *getlogin()* function need not be reentrant. A function that is not required to be reentrant is
 17137 not required to be thread-safe.

17138 TSF The *getlogin_r()* function shall put the name associated by the login activity with the controlling
 17139 terminal of the current process in the character array pointed to by *name*. The array is *namesize*
 17140 characters long and should have space for the name and the terminating null character. The
 17141 maximum size of the login name is {LOGIN_NAME_MAX}.

17142 If *getlogin_r()* is successful, *name* points to the name the user used at login, even if there are
 17143 several login names with the same user ID.

17144 **RETURN VALUE**

17145 Upon successful completion, *getlogin()* shall return a pointer to the login name or a null pointer
 17146 if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to
 17147 indicate the error.

17148 The return value from *getlogin()* may point to static data whose content is overwritten by each
 17149 call.

17150 TSF If successful, the *getlogin_r()* function shall return zero; otherwise, an error number shall be
 17151 returned to indicate the error.

17152 **ERRORS**17153 The *getlogin()* and *getlogin_r()* functions may fail if:

17154 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

17155 [ENFILE] The maximum allowable number of files is currently open in the system.

17156 [ENXIO] The calling process has no controlling terminal.

17157 The *getlogin_r()* function may fail if:

17158 TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned
 17159 including the terminating null character.

17160 **EXAMPLES**17161 **Getting the User Login Name**

17162 The following example calls the *getlogin()* function to obtain the name of the user associated
 17163 with the calling process, and passes this information to the *getpwnam()* function to get the
 17164 associated user database information.

```

17165 #include <unistd.h>
17166 #include <sys/types.h>
17167 #include <pwd.h>
17168 #include <stdio.h>
17169 ...
17170 char *lgn;
17171 struct passwd *pw;
17172 ...
17173 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
17174     fprintf(stderr, "Get of user information failed.\n"); exit(1);
17175 }
```

17176 **APPLICATION USAGE**

17177 Three names associated with the current process can be determined: *getpwuid(geteuid())* shall
 17178 return the name associated with the effective user ID of the process; *getlogin()* shall return the
 17179 name associated with the current login activity; and *getpwuid(getuid())* shall return the name
 17180 associated with the real user ID of the process.

17181 The *getlogin_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 17182 possibly using a static data area that may be overwritten by each call.

17183 **RATIONALE**

17184 The *getlogin()* function returns a pointer to the user's login name. The same user ID may be
 17185 shared by several login names. If it is desired to get the user database entry that is used during
 17186 login, the result of *getlogin()* should be used to provide the argument to the *getpwnam()*
 17187 function. (This might be used to determine the user's login shell, particularly where a single user
 17188 has multiple login shells with distinct login names, but the same user ID.)

17189 The information provided by the *cuserid()* function, which was originally defined in the
 17190 POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
17191 getpwuid(geteuid())
```

17192 while the information provided by historical implementations of *cuserid()* can be obtained by:

```
17193 getpwuid(getuid())
```

17194 The thread-safe version of this function places the user name in a user-supplied buffer and
 17195 returns a non-zero value if it fails. The non-thread-safe version may return the name in a static
 17196 data area that may be overwritten by each call.

17197 **FUTURE DIRECTIONS**

17198 None.

17199 **SEE ALSO**

17200 *getpwnam()*, *getpwuid()*, *geteuid()*, *getuid()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 17201 **<limits.h>**, **<unistd.h>**

17202 **CHANGE HISTORY**

17203 First released in Issue 1. Derived from System V Release 2.0.

17204 **Issue 5**

17205 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
17206 VALUE section.

17207 The *getlogin_r()* function is included for alignment with the POSIX Threads Extension.

17208 A note indicating that the *getlogin()* function need not be reentrant is added to the
17209 DESCRIPTION.

17210 **Issue 6**

17211 The *getlogin_r()* function is marked as part of the Thread-Safe Functions option.

17212 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

17213 The following new requirements on POSIX implementations derive from alignment with the
17214 Single UNIX Specification:

17215 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

17216 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

17217 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
17218 its avoidance of possibly using a static data area.

17219 **NAME**17220 getmsg, getpmsg — receive next message from a STREAMS file (**STREAMS**)17221 **SYNOPSIS**17222 XSR `#include <stropts.h>`

```
17223 int getmsg(int fildes, struct strbuf *restrict ctlptr,
17224           struct strbuf *restrict dataptr, int *restrict flagsp);
17225 int getpmsg(int fildes, struct strbuf *restrict ctlptr,
17226            struct strbuf *restrict dataptr, int *restrict bandp,
17227            int *restrict flagsp);
```

17229 **DESCRIPTION**

17230 The *getmsg()* function shall retrieve the contents of a message located at the head of the
 17231 STREAM head read queue associated with a STREAMS file and place the contents into one or
 17232 more buffers. The message contains either a data part, a control part, or both. The data and
 17233 control parts of the message shall be placed into separate buffers, as described below. The
 17234 semantics of each part are defined by the originator of the message.

17235 The *getpmsg()* function shall be equivalent to *getmsg()*, except that it provides finer control over
 17236 the priority of the messages received. Except where noted, all requirements on *getmsg()* also
 17237 pertain to *getpmsg()*.

17238 The *fildes* argument specifies a file descriptor referencing a STREAMS-based file.

17239 The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points
 17240 to a buffer in which the data or control information is to be placed, and the *maxlen* member
 17241 indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall
 17242 contain the number of bytes of data or control information actually received. The *len* member
 17243 shall be set to 0 if there is a zero-length control or data part and *len* shall be set to -1 if no data or
 17244 control information is present in the message.

17245 When *getmsg()* is called, *flagsp* should point to an integer that indicates the type of message the
 17246 process is able to receive. This is described further below.

17247 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold
 17248 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is -1, the
 17249 control (or data) part of the message shall not be processed and shall be left on the STREAM
 17250 head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to -1. If the
 17251 *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part
 17252 shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0
 17253 and there are more than 0 bytes of control (or data) information, that information shall be left on
 17254 the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the
 17255 control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder
 17256 of the message shall be left on the STREAM head read queue and a non-zero return value shall
 17257 be provided.

17258 By default, *getmsg()* shall process the first available message on the STREAM head read queue.
 17259 However, a process may choose to retrieve only high-priority messages by setting the integer
 17260 pointed to by *flagsp* to RS_HIPRI. In this case, *getmsg()* shall only process the next message if it is
 17261 a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall
 17262 be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to RS_HIPRI if a
 17263 high-priority message was retrieved, or 0 otherwise.

17264 For *getpmsg()*, the flags are different. The *flagsp* argument points to a bitmask with the following
 17265 mutually-exclusive flags defined: MSG_HIPRI, MSG_BAND, and MSG_ANY. Like *getmsg()*,

17266 *getpmsg()* shall process the first available message on the STREAM head read queue. A process
 17267 may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to
 17268 MSG_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg()* shall only process
 17269 the next message if it is a high-priority message. In a similar manner, a process may choose to
 17270 retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to
 17271 MSG_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case,
 17272 *getpmsg()* shall only process the next message if it is in a priority band equal to, or greater than,
 17273 the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the
 17274 first message off the queue, the integer pointed to by *flagsp* should be set to MSG_ANY and the
 17275 integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high-
 17276 priority message, the integer pointed to by *flagsp* shall be set to MSG_HIPRI and the integer
 17277 pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to
 17278 MSG_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

17279 If O_NONBLOCK is not set, *getmsg()* and *getpmsg()* shall block until a message of the type
 17280 specified by *flagsp* is available at the front of the STREAM head read queue. If O_NONBLOCK is
 17281 set and a message of the specified type is not present at the front of the read queue, *getmsg()* and
 17282 *getpmsg()* shall fail and set *errno* to [EAGAIN].

17283 If a hangup occurs on the STREAM from which messages are retrieved, *getmsg()* and *getpmsg()*
 17284 shall continue to operate normally, as described above, until the STREAM head read queue is
 17285 empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

17286 RETURN VALUE

17287 Upon successful completion, *getmsg()* and *getpmsg()* shall return a non-negative value. A value
 17288 of 0 indicates that a full message was read successfully. A return value of MORECTL indicates
 17289 that more control information is waiting for retrieval. A return value of MOREDATA indicates
 17290 that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL
 17291 and MOREDATA indicates that both types of information remain. Subsequent *getmsg()* and
 17292 *getpmsg()* calls shall retrieve the remainder of the message. However, if a message of higher
 17293 priority has come in on the STREAM head read queue, the next call to *getmsg()* or *getpmsg()*
 17294 shall retrieve that higher-priority message before retrieving the remainder of the previous
 17295 message.

17296 If the high priority control part of the message is consumed, the message shall be placed back on
 17297 the queue as a normal message of band 0. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve
 17298 the remainder of the message. If, however, a priority message arrives or already exists on the
 17299 STREAM head, the subsequent call to *getmsg()* or *getpmsg()* shall retrieve the higher-priority
 17300 message before retrieving the remainder of the message that was put back.

17301 Upon failure, *getmsg()* and *getpmsg()* shall return -1 and set *errno* to indicate the error.

17302 ERRORS

17303 The *getmsg()* and *getpmsg()* functions shall fail if:

17304	[EAGAIN]	The O_NONBLOCK flag is set and no messages are available.
17305	[EBADF]	The <i>fildes</i> argument is not a valid file descriptor open for reading.
17306	[EBADMSG]	The queued message to be read is not valid for <i>getmsg()</i> or <i>getpmsg()</i> or a 17307 pending file descriptor is at the STREAM head.
17308	[EINTR]	A signal was caught during <i>getmsg()</i> or <i>getpmsg()</i> .
17309	[EINVAL]	An illegal value was specified by <i>flagsp</i> , or the STREAM or multiplexer 17310 referenced by <i>fildes</i> is linked (directly or indirectly) downstream from a 17311 multiplexer.

17312 [ENOSTR] A STREAM is not associated with *fildev*.

17313 In addition, *getmsg()* and *getpmsg()* shall fail if the STREAM head had processed an
17314 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of
17315 *getmsg()* or *getpmsg()* but reflects the prior error.

17316 EXAMPLES

17317 Getting Any Message

17318 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call
17319 to *getmsg()* retrieves any available message on the associated STREAM-head read queue,
17320 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,
17321 respectively.

```
17322 #include <stropts.h>
17323 ...
17324 int fd;
17325 char ctrlbuf[128];
17326 char databuf[512];
17327 struct strbuf ctrl;
17328 struct strbuf data;
17329 int flags = 0;
17330 int ret;

17331 ctrl.buf = ctrlbuf;
17332 ctrl.maxlen = sizeof(ctrlbuf);

17333 data.buf = databuf;
17334 data.maxlen = sizeof(databuf);

17335 ret = getmsg (fd, &ctrl, &data, &flags);
```

17336 Getting the First Message off the Queue

17337 In the following example, the call to *getpmsg()* retrieves the first available message on the
17338 associated STREAM-head read queue.

```
17339 #include <stropts.h>
17340 ...

17341 int fd;
17342 char ctrlbuf[128];
17343 char databuf[512];
17344 struct strbuf ctrl;
17345 struct strbuf data;
17346 int band = 0;
17347 int flags = MSG_ANY;
17348 int ret;

17349 ctrl.buf = ctrlbuf;
17350 ctrl.maxlen = sizeof(ctrlbuf);

17351 data.buf = databuf;
17352 data.maxlen = sizeof(databuf);

17353 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

17354 **APPLICATION USAGE**

17355 None.

17356 **RATIONALE**

17357 None.

17358 **FUTURE DIRECTIONS**

17359 None.

17360 **SEE ALSO**

17361 Section 2.6 (on page 38), *poll()*, *putmsg()*, *read()*, *write()*, the Base Definitions volume of
17362 IEEE Std 1003.1-2001, <**stropts.h**>

17363 **CHANGE HISTORY**

17364 First released in Issue 4, Version 2.

17365 **Issue 5**

17366 Moved from X/OPEN UNIX extension to BASE.

17367 A paragraph regarding “high-priority control parts of messages” is added to the RETURN
17368 VALUE section.

17369 **Issue 6**

17370 This function is marked as part of the XSI STREAMS Option Group.

17371 The **restrict** keyword is added to the *getmsg()* and *getpmsg()* prototypes for alignment with the
17372 ISO/IEC 9899:1999 standard.

17373 **NAME**

17374 getnameinfo — get name information

17375 **SYNOPSIS**

17376 #include <sys/socket.h>

17377 #include <netdb.h>

```
17378 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
17379 char *restrict node, socklen_t nodelen, char *restrict service,
17380 socklen_t servicelen, int flags);
```

17381 **DESCRIPTION**

17382 The *getnameinfo()* function shall translate a socket address to a node name and service location,
 17383 all of which are defined as in *getaddrinfo()*.

17384 The *sa* argument points to a socket address structure to be translated.

17385 **IPv6** If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible
 17386 IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node
 17387 name for that IPv4 address.

17388 **Note:** The IPv6 unspecified address ("::") and the IPv6 loopback address "::1") are not IPv4-
 17389 compatible addresses. If the address is the IPv6 unspecified address ("::"), a lookup is not
 17390 performed, and the [EAI_NONAME] error is returned.

17391 If the *node* argument is non-NULL and the *nodelen* argument is non-zero, then the *node* argument
 17392 points to a buffer able to contain up to *nodelen* characters that receives the node name as a null-
 17393 terminated string. If the *node* argument is NULL or the *nodelen* argument is zero, the node name
 17394 shall not be returned. If the node's name cannot be located, the numeric form of the address
 17395 contained in the socket address structure pointed to by the *sa* argument is returned instead of its
 17396 name.

17397 If the *service* argument is non-NULL and the *servicelen* argument is non-zero, then the *service*
 17398 argument points to a buffer able to contain up to *servicelen* bytes that receives the service name
 17399 as a null-terminated string. If the *service* argument is NULL or the *servicelen* argument is zero,
 17400 the service name shall not be returned. If the service's name cannot be located, the numeric form
 17401 of the service address (for example, its port number) shall be returned instead of its name.

17402 The *flags* argument is a flag that changes the default actions of the function. By default the fully-
 17403 qualified domain name (FQDN) for the host shall be returned, but:

- 17404 • If the flag bit NI_NOFQDN is set, only the node name portion of the FQDN shall be returned
 17405 for local hosts.
- 17406 • If the flag bit NI_NUMERICHOST is set, the numeric form of the address contained in the
 17407 socket address structure pointed to by the *sa* argument shall be returned instead of its name,
 17408 under all circumstances.
- 17409 • If the flag bit NI_NAMEREQD is set, an error shall be returned if the host's name cannot be
 17410 located.
- 17411 • If the flag bit NI_NUMERICSERV is set, the numeric form of the service address shall be
 17412 returned (for example, its port number) instead of its name, under all circumstances.
- 17413 • If the flag bit NI_NUMERICSERVICE is set, the numeric form of the scope identifier shall be
 17414 returned (for example, interface index) instead of its name. This flag shall be ignored if the *sa*
 17415 argument is not an IPv6 address.
- 17416 • If the flag bit NI_DGRAM is set, this indicates that the service is a datagram service
 17417 (SOCK_DGRAM). The default behavior shall assume that the service is a stream service

17418 (SOCK_STREAM).

17419 **Notes:**

- 17420 1. The two NI_NUMERICxxx flags are required to support the **-n** flag that many
- 17421 commands provide.
- 17422 2. The NI_DGRAM flag is required for the few AF_INET and AF_INET6 port numbers (for
- 17423 example, [512,514]) that represent different services for UDP and TCP.

17424 The *getnameinfo()* function shall be thread-safe.

17425 RETURN VALUE

17426 A zero return value for *getnameinfo()* indicates successful completion; a non-zero return value

17427 indicates failure. The possible values for the failures are listed in the ERRORS section.

17428 Upon successful completion, *getnameinfo()* shall return the *node* and *service* names, if requested,

17429 in the buffers provided. The returned names are always null-terminated strings.

17430 ERRORS

17431 The *getnameinfo()* function shall fail and return the corresponding value if:

17432 [EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

17433 [EAI_BADFLAGS]

17434 The *flags* had an invalid value.

17435 [EAI_FAIL] A non-recoverable error occurred.

17436 [EAI_FAMILY] The address family was not recognized or the address length was invalid for

17437 the specified family.

17438 [EAI_MEMORY] There was a memory allocation failure.

17439 [EAI_NONAME] The name does not resolve for the supplied parameters.

17440 NI_NAMEREQD is set and the host's name cannot be located, or both

17441 *nodename* and *servname* were null.

17442 [EAI_OVERFLOW]

17443 An argument buffer overflowed. The buffer pointed to by the *node* argument

17444 or the *service* argument was too small.

17445 [EAI_SYSTEM] A system error occurred. The error code can be found in *errno*.

17446 EXAMPLES

17447 None.

17448 APPLICATION USAGE

17449 If the returned values are to be used as part of any further name resolution (for example, passed

17450 to *getaddrinfo()*), applications should provide buffers large enough to store any result possible

17451 on the system.

17452 Given the IPv4-mapped IPv6 address "::

17453 lookup as if the socket address structure contains the IPv4 address "1.2.3.4".

17454 RATIONALE

17455 None.

17456 FUTURE DIRECTIONS

17457 None.

17458 **SEE ALSO**

17459 *gai_strerror()*, *getaddrinfo()*, *getservbyname()*, *inet_ntop()*, *socket()*, the Base Definitions volume of
17460 IEEE Std 1003.1-2001, <**netdb.h**>, <**sys/socket.h**>

17461 **CHANGE HISTORY**

17462 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17463 The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the
17464 ISO/IEC 9899:1999 standard.

17465 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in |
17466 the SYNOPSIS and DESCRIPTION for alignment with IPv6. |

17467 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the |
17468 [EAI_OVERFLOW] error to the ERRORS section. |

17469 **NAME**

17470 getnetbyaddr, getnetbyname, getnetent — network database functions

17471 **SYNOPSIS**

17472 #include <netdb.h>

17473 struct netent *getnetbyaddr(uint32_t net, int type);

17474 struct netent *getnetbyname(const char *name);

17475 struct netent *getnetent(void);

17476 **DESCRIPTION**

17477 Refer to *endnetent()*.

17478 **NAME**

17479 getopt, optarg, opterr, optind, optopt — command option parsing

17480 **SYNOPSIS**

17481 #include <unistd.h>

17482 int getopt(int argc, char * const argv[], const char *optstring);

17483 extern char *optarg;

17484 extern int optind, opterr, optopt;

17485 **DESCRIPTION**

17486 The *getopt()* function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5,
 17487 6, 7, 9, and 10 in the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax
 17488 Guidelines.

17489 The parameters *argc* and *argv* are the argument count and argument array as passed to *main()*
 17490 (see *exec*). The argument *optstring* is a string of recognized option characters; if a character is
 17491 followed by a colon, the option takes an argument. All option characters allowed by Utility
 17492 Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as
 17493 an extension.

17494 The variable *optind* is the index of the next element of the *argv[]* vector to be processed. It shall
 17495 be initialized to 1 by the system, and *getopt()* shall update it when it finishes with each element
 17496 of *argv[]*. When an element of *argv[]* contains multiple option characters, it is unspecified how
 17497 *getopt()* determines which options have already been processed.

17498 The *getopt()* function shall return the next option character (if one is found) from *argv* that
 17499 matches a character in *optstring*, if there is one that matches. If the option takes an argument,
 17500 *getopt()* shall set the variable *optarg* to point to the option-argument as follows:

- 17501 1. If the option was the last character in the string pointed to by an element of *argv*, then
 17502 *optarg* shall contain the next element of *argv*, and *optind* shall be incremented by 2. If the
 17503 resulting value of *optind* is greater than *argc*, this indicates a missing option-argument, and
 17504 *getopt()* shall return an error indication.
- 17505 2. Otherwise, *optarg* shall point to the string following the option character in that element of
 17506 *argv*, and *optind* shall be incremented by 1.

17507 If, when *getopt()* is called:

17508 *argv[optind]* is a null pointer
 17509 **argv[optind]* is not the character *-*
 17510 *argv[optind]* points to the string *--*

17511 *getopt()* shall return *-1* without changing *optind*. If:

17512 *argv[optind]* points to the string *--*

17513 *getopt()* shall return *-1* after incrementing *optind*.

17514 If *getopt()* encounters an option character that is not contained in *optstring*, it shall return the
 17515 question-mark (*' ? '*) character. If it detects a missing option-argument, it shall return the colon
 17516 character (*' : '*) if the first character of *optstring* was a colon, or a question-mark character (*' ? '*)
 17517 otherwise. In either case, *getopt()* shall set the variable *optopt* to the option character that caused
 17518 the error. If the application has not set the variable *opterr* to 0 and the first character of *optstring*
 17519 is not a colon, *getopt()* shall also print a diagnostic message to *stderr* in the format specified for
 17520 the *getopts* utility.

17521 The *getopt()* function need not be reentrant. A function that is not required to be reentrant is not
 17522 required to be thread-safe.

17523 **RETURN VALUE**

17524 The *getopt()* function shall return the next option character specified on the command line.

17525 A colon (':') shall be returned if *getopt()* detects a missing argument and the first character of
17526 *optstring* was a colon (':').

17527 A question mark ('?') shall be returned if *getopt()* encounters an option character not in
17528 *optstring* or detects a missing argument and the first character of *optstring* was not a colon (':').

17529 Otherwise, *getopt()* shall return -1 when all command line options are parsed.

17530 **ERRORS**

17531 No errors are defined.

17532 **EXAMPLES**17533 **Parsing Command Line Options**

17534 The following code fragment shows how you might process the arguments for a utility that can
17535 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require
17536 arguments:

```
17537 #include <unistd.h>
17538
17539 int
17540 main(int argc, char *argv[ ])
17541 {
17542     int c;
17543     int bflg, aflag, errflag;
17544     char *ifile;
17545     char *ofile;
17546     extern char *optarg;
17547     extern int optind, optopt;
17548     . . .
17549     while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
17550         switch(c) {
17551             case 'a':
17552                 if (bflg)
17553                     errflag++;
17554                 else
17555                     aflag++;
17556                 break;
17557             case 'b':
17558                 if (aflag)
17559                     errflag++;
17560                 else {
17561                     bflg++;
17562                     bproc();
17563                 }
17564                 break;
17565             case 'f':
17566                 ifile = optarg;
17567                 break;
17568             case 'o':
17569                 ofile = optarg;
17570                 break;
```

```

17570         case ':':          /* -f or -o without operand */
17571             fprintf(stderr,
17572                 "Option -%c requires an operand\n", optopt);
17573             errflg++;
17574             break;
17575         case '?':
17576             fprintf(stderr,
17577                 "Unrecognized option: -%c\n", optopt);
17578             errflg++;
17579     }
17580 }
17581 if (errflg) {
17582     fprintf(stderr, "usage: . . . ");
17583     exit(2);
17584 }
17585 for ( ; optind < argc; optind++) {
17586     if (access(argv[optind], R_OK)) {
17587         . . .
17588     }

```

17589 This code accepts any of the following as equivalent:

```

17590 cmd -ao arg path path
17591 cmd -a -o arg path path
17592 cmd -o arg -a path path
17593 cmd -a -o arg -- path path
17594 cmd -a -oarg path path
17595 cmd -aoarg path path

```

17596 **Checking Options and Arguments**

17597 The following example parses a set of command line options and prints messages to standard
 17598 output for each option and argument that it encounters.

```

17599 #include <unistd.h>
17600 #include <stdio.h>
17601 ...
17602 int c;
17603 char *filename;
17604 extern char *optarg;
17605 extern int optind, optopt, opterr;
17606 ...
17607 while ((c = getopt(argc, argv, ":abf:")) != -1) {
17608     switch(c) {
17609         case 'a':
17610             printf("a is set\n");
17611             break;
17612         case 'b':
17613             printf("b is set\n");
17614             break;
17615         case 'f':
17616             filename = optarg;
17617             printf("filename is %s\n", filename);
17618             break;

```

```

17619         case ':':
17620             printf("-%c without filename\n", optopt);
17621             break;
17622         case '?':
17623             printf("unknown arg %c\n", optopt);
17624             break;
17625     }
17626 }

```

17627 **Selecting Options from the Command Line**

17628 The following example selects the type of database routines the user wants to use based on the
 17629 *Options* argument.

```

17630 #include <unistd.h>
17631 #include <string.h>
17632 ...
17633 char *Options = "hdbtl";
17634 ...
17635 int dbtype, i;
17636 char c;
17637 char *st;
17638 ...
17639 dbtype = 0;
17640 while ((c = getopt(argc, argv, Options)) != -1) {
17641     if ((st = strchr(Options, c)) != NULL) {
17642         dbtype = st - Options;
17643         break;
17644     }
17645 }

```

17646 **APPLICATION USAGE**

17647 The *getopt()* function is only required to support option characters included in Utility Syntax
 17648 Guideline 3. Many historical implementations of *getopt()* support other characters as options.
 17649 This is an allowed extension, but applications that use extensions are not maximally portable.
 17650 Note that support for multi-byte option characters is only possible when such characters can be
 17651 represented as type **int**.

17652 **RATIONALE**

17653 The *optopt* variable represents historical practice and allows the application to obtain the identity
 17654 of the invalid option.

17655 The description has been written to make it clear that *getopt()*, like the *getopts* utility, deals with
 17656 option-arguments whether separated from the option by <blank>s or not. Note that the
 17657 requirements on *getopt()* and *getopts* are more stringent than the Utility Syntax Guidelines.

17658 The *getopt()* function shall return **-1**, rather than EOF, so that **<stdio.h>** is not required.

17659 The special significance of a colon as the first character of *optstring* makes *getopt()* consistent
 17660 with the *getopts* utility. It allows an application to make a distinction between a missing
 17661 argument and an incorrect option letter without having to examine the option letter. It is true
 17662 that a missing argument can only be detected in one case, but that is a case that has to be
 17663 considered.

17664 **FUTURE DIRECTIONS**

17665 None.

17666 **SEE ALSO**

17667 *exec*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>, the Shell and Utilities
17668 volume of IEEE Std 1003.1-2001

17669 **CHANGE HISTORY**

17670 First released in Issue 1. Derived from Issue 1 of the SVID.

17671 **Issue 5**17672 A note indicating that the *getopt()* function need not be reentrant is added to the DESCRIPTION.17673 **Issue 6**

17674 IEEE PASC Interpretation 1003.2 #150 is applied.

17675 **NAME**

17676 getpeername — get the name of the peer socket

17677 **SYNOPSIS**

17678 #include <sys/socket.h>

17679 int getpeername(int *socket*, struct sockaddr *restrict *address*,
17680 socklen_t *restrict *address_len*);

17681 **DESCRIPTION**

17682 The *getpeername()* function shall retrieve the peer address of the specified socket, store this
17683 address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this
17684 address in the object pointed to by the *address_len* argument.

17685 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
17686 the stored address shall be truncated.

17687 If the protocol permits connections by unbound clients, and the peer is not bound, then the value
17688 stored in the object pointed to by *address* is unspecified.

17689 **RETURN VALUE**

17690 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
17691 indicate the error.

17692 **ERRORS**

17693 The *getpeername()* function shall fail if:

17694 [EBADF] The *socket* argument is not a valid file descriptor.

17695 [EINVAL] The socket has been shut down.

17696 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.

17697 [ENOTSOCK] The *socket* argument does not refer to a socket.

17698 [EOPNOTSUPP] The operation is not supported for the socket protocol.

17699 The *getpeername()* function may fail if:

17700 [ENOBUFS] Insufficient resources were available in the system to complete the call.

17701 **EXAMPLES**

17702 None.

17703 **APPLICATION USAGE**

17704 None.

17705 **RATIONALE**

17706 None.

17707 **FUTURE DIRECTIONS**

17708 None.

17709 **SEE ALSO**

17710 *accept()*, *bind()*, *getsockname()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,
17711 <sys/socket.h>

17712 **CHANGE HISTORY**

17713 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17714 The **restrict** keyword is added to the *getpeername()* prototype for alignment with the
17715 ISO/IEC 9899:1999 standard.

17716 **NAME**

17717 getpgid — get the process group ID for a process

17718 **SYNOPSIS**

17719 xSI #include <unistd.h>

17720 pid_t getpgid(pid_t pid);

17721

17722 **DESCRIPTION**

17723 The *getpgid()* function shall return the process group ID of the process whose process ID is equal
 17724 to *pid*. If *pid* is equal to 0, *getpgid()* shall return the process group ID of the calling process.

17725 **RETURN VALUE**

17726 Upon successful completion, *getpgid()* shall return a process group ID. Otherwise, it shall return
 17727 (**pid_t**)−1 and set *errno* to indicate the error.

17728 **ERRORS**17729 The *getpgid()* function shall fail if:

17730 [EPERM] The process whose process ID is equal to *pid* is not in the same session as the
 17731 calling process, and the implementation does not allow access to the process
 17732 group ID of that process from the calling process.

17733 [ESRCH] There is no process with a process ID equal to *pid*.

17734 The *getpgid()* function may fail if:

17735 [EINVAL] The value of the *pid* argument is invalid.

17736 **EXAMPLES**

17737 None.

17738 **APPLICATION USAGE**

17739 None.

17740 **RATIONALE**

17741 None.

17742 **FUTURE DIRECTIONS**

17743 None.

17744 **SEE ALSO**

17745 *exec*, *fork()*, *getpgrp()*, *getpid()*, *getsid()*, *setpgid()*, *setsid()*, the Base Definitions volume of
 17746 IEEE Std 1003.1-2001, <**unistd.h**>

17747 **CHANGE HISTORY**

17748 First released in Issue 4, Version 2.

17749 **Issue 5**

17750 Moved from X/OPEN UNIX extension to BASE.

17751 **NAME**

17752 getpgrp — get the process group ID of the calling process

17753 **SYNOPSIS**

17754 #include <unistd.h>

17755 pid_t getpgrp(void);

17756 **DESCRIPTION**

17757 The *getpgrp()* function shall return the process group ID of the calling process.

17758 **RETURN VALUE**

17759 The *getpgrp()* function shall always be successful and no return value is reserved to indicate an error.

17761 **ERRORS**

17762 No errors are defined.

17763 **EXAMPLES**

17764 None.

17765 **APPLICATION USAGE**

17766 None.

17767 **RATIONALE**

17768 4.3 BSD provides a *getpgrp()* function that returns the process group ID for a specified process.
17769 Although this function supports job control, all known job control shells always specify the
17770 calling process with this function. Thus, the simpler System V *getpgrp()* suffices, and the added
17771 complexity of the 4.3 BSD *getpgrp()* is provided by the XSI extension *getpgid()*.

17772 **FUTURE DIRECTIONS**

17773 None.

17774 **SEE ALSO**

17775 *exec*, *fork()*, *getpgid()*, *getpid()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of
17776 IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

17777 **CHANGE HISTORY**

17778 First released in Issue 1. Derived from Issue 1 of the SVID.

17779 **Issue 6**

17780 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

17781 The following new requirements on POSIX implementations derive from alignment with the
17782 Single UNIX Specification:

- 17783
 - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
17784 required for conforming implementations of previous POSIX specifications, it was not
17785 required for UNIX applications.

17786 **NAME**

17787 `getpid` — get the process ID

17788 **SYNOPSIS**

17789 `#include <unistd.h>`

17790 `pid_t getpid(void);`

17791 **DESCRIPTION**

17792 The `getpid()` function shall return the process ID of the calling process.

17793 **RETURN VALUE**

17794 The `getpid()` function shall always be successful and no return value is reserved to indicate an
17795 error.

17796 **ERRORS**

17797 No errors are defined.

17798 **EXAMPLES**

17799 None.

17800 **APPLICATION USAGE**

17801 None.

17802 **RATIONALE**

17803 None.

17804 **FUTURE DIRECTIONS**

17805 None.

17806 **SEE ALSO**

17807 `exec`, `fork()`, `getpgrp()`, `getppid()`, `kill()`, `setpgid()`, `setsid()`, the Base Definitions volume of
17808 IEEE Std 1003.1-2001, `<sys/types.h>`, `<unistd.h>`

17809 **CHANGE HISTORY**

17810 First released in Issue 1. Derived from Issue 1 of the SVID.

17811 **Issue 6**

17812 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

17813 The following new requirements on POSIX implementations derive from alignment with the
17814 Single UNIX Specification:

- 17815 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
17816 required for conforming implementations of previous POSIX specifications, it was not
17817 required for UNIX applications.

17818 **NAME**

17819 getpmsg — receive next message from a STREAMS file

17820 **SYNOPSIS**

17821 xSI #include <stropts.h>

```
17822       int getpmsg(int fildev, struct strbuf *restrict ctlptr,  
17823                struct strbuf *restrict dataptr, int *restrict bandp,  
17824                int *restrict flagsp);
```

17825

17826 **DESCRIPTION**17827 Refer to *getmsg()*.

17828 **NAME**

17829 getppid — get the parent process ID

17830 **SYNOPSIS**

17831 #include <unistd.h>

17832 pid_t getppid(void);

17833 **DESCRIPTION**

17834 The *getppid()* function shall return the parent process ID of the calling process.

17835 **RETURN VALUE**

17836 The *getppid()* function shall always be successful and no return value is reserved to indicate an
17837 error.

17838 **ERRORS**

17839 No errors are defined.

17840 **EXAMPLES**

17841 None.

17842 **APPLICATION USAGE**

17843 None.

17844 **RATIONALE**

17845 None.

17846 **FUTURE DIRECTIONS**

17847 None.

17848 **SEE ALSO**

17849 *exec*, *fork()*, *getpgid()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of
17850 IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

17851 **CHANGE HISTORY**

17852 First released in Issue 1. Derived from Issue 1 of the SVID.

17853 **Issue 6**

17854 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

17855 The following new requirements on POSIX implementations derive from alignment with the
17856 Single UNIX Specification:

- 17857
 - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
17858 required for conforming implementations of previous POSIX specifications, it was not
17859 required for UNIX applications.

17860 **NAME**

17861 getpriority, setpriority — get and set the nice value

17862 **SYNOPSIS**

17863 xSI #include <sys/resource.h>

17864 int getpriority(int which, id_t who);

17865 int setpriority(int which, id_t who, int value);

17866

17867 **DESCRIPTION**

17868 The *getpriority()* function shall obtain the nice value of a process, process group, or user. The
 17869 *setpriority()* function shall set the nice value of a process, process group, or user to
 17870 *value*+{NZERO}.

17871 Target processes are specified by the values of the *which* and *who* arguments. The *which*
 17872 argument may be one of the following values: PRIO_PROCESS, PRIO_PGRP, or PRIO_USER,
 17873 indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an
 17874 effective user ID, respectively. A 0 value for the *who* argument specifies the current process,
 17875 process group, or user.

17876 The nice value set with *setpriority()* shall be applied to the process. If the process is multi-
 17877 threaded, the nice value shall affect all system scope threads in the process.

17878 If more than one process is specified, *getpriority()* shall return value {NZERO} less than the
 17879 lowest nice value pertaining to any of the specified processes, and *setpriority()* shall set the nice
 17880 values of all of the specified processes to *value*+{NZERO}.

17881 The default nice value is {NZERO}; lower nice values shall cause more favorable scheduling.
 17882 While the range of valid nice values is [0,{NZERO}*2-1], implementations may enforce more
 17883 restrictive limits. If *value*+{NZERO} is less than the system's lowest supported nice value,
 17884 *setpriority()* shall set the nice value to the lowest supported value; if *value*+{NZERO} is greater
 17885 than the system's highest supported nice value, *setpriority()* shall set the nice value to the highest
 17886 supported value.

17887 Only a process with appropriate privileges can lower its nice value.

17888 PS|TPS Any processes or threads using SCHED_FIFO or SCHED_RR shall be unaffected by a call to
 17889 *setpriority()*. This is not considered an error. A process which subsequently reverts to
 17890 SCHED_OTHER need not have its priority affected by such a *setpriority()* call.

17891 The effect of changing the nice value may vary depending on the process-scheduling algorithm
 17892 in effect.

17893 Since *getpriority()* can return the value -1 on successful completion, it is necessary to set *errno* to
 17894 0 prior to a call to *getpriority()*. If *getpriority()* returns the value -1, then *errno* can be checked to
 17895 see if an error occurred or if the value is a legitimate nice value.

17896 **RETURN VALUE**

17897 Upon successful completion, *getpriority()* shall return an integer in the range -{NZERO} to
 17898 {NZERO}-1. Otherwise, -1 shall be returned and *errno* set to indicate the error.

17899 Upon successful completion, *setpriority()* shall return 0; otherwise, -1 shall be returned and *errno*
 17900 set to indicate the error.

17901 **ERRORS**

17902 The *getpriority()* and *setpriority()* functions shall fail if:

17903 [ESRCH] No process could be located using the *which* and *who* argument values
 17904 specified.

17905 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who*
 17906 argument is not a valid process ID, process group ID, or user ID.

17907 In addition, *setpriority()* may fail if:

17908 [EPERM] A process was located, but neither the real nor effective user ID of the
 17909 executing process match the effective user ID of the process whose nice value
 17910 is being changed.

17911 [EACCES] A request was made to change the nice value to a lower numeric value and
 17912 the current process does not have appropriate privileges.

17913 EXAMPLES

17914 Using *getpriority()*

17915 The following example returns the current scheduling priority for the process ID returned by the
 17916 call to *getpid()*.

```
17917 #include <sys/resource.h>
17918 ...
17919 int which = PRIO_PROCESS;
17920 id_t pid;
17921 int ret;

17922 pid = getpid();
17923 ret = getpriority(which, pid);
```

17924 Using *setpriority()*

17925 The following example sets the priority for the current process ID to -20.

```
17926 #include <sys/resource.h>
17927 ...
17928 int which = PRIO_PROCESS;
17929 id_t pid;
17930 int priority = -20;
17931 int ret;

17932 pid = getpid();
17933 ret = setpriority(which, pid, priority);
```

17934 APPLICATION USAGE

17935 The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value
 17936 $-\{\text{NZERO}\}$). The nice value is in the range $[0, 2*\{\text{NZERO}\} - 1]$, while the return value for
 17937 *getpriority()* and the third parameter for *setpriority()* are in the range $[-\{\text{NZERO}\}, \{\text{NZERO}\} - 1]$.

17938 RATIONALE

17939 None.

17940 FUTURE DIRECTIONS

17941 None.

17942 SEE ALSO

17943 *nice()*, *sched_get_priority_max()*, *sched_setscheduler()*, the Base Definitions volume of
 17944 IEEE Std 1003.1-2001, **<sys/resource.h>**

17945 **CHANGE HISTORY**

17946 First released in Issue 4, Version 2.

17947 **Issue 5**

17948 Moved from X/OPEN UNIX extension to BASE.

17949 The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion
17950 with functionality in the POSIX Realtime Extension.

17951 NAME

17952 getprotobyname, getprotobynumber, getprotent — network protocol database functions

17953 SYNOPSIS

17954 #include <netdb.h>

17955 struct protoent *getprotobyname(const char *name);

17956 struct protoent *getprotobynumber(int proto);

17957 struct protoent *getprotoent(void);

17958 DESCRIPTION

17959 Refer to *endprotoent()*.

17960 **NAME**

17961 getpwent — get user database entry

17962 **SYNOPSIS**

17963 xSI #include <pwd.h>

17964 struct passwd *getpwent(void);

17965

17966 **DESCRIPTION**17967 Refer to *endpwent()*.

17968 **NAME**

17969 getpwnam, getpwnam_r — search user database for a name

17970 **SYNOPSIS**

17971 #include <pwd.h>

17972 struct passwd *getpwnam(const char *name);

17973 TSF int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,

17974 size_t bufsize, struct passwd **result);

17975

17976 **DESCRIPTION**17977 The *getpwnam()* function shall search the user database for an entry with a matching *name*.17978 The *getpwnam()* function need not be reentrant. A function that is not required to be reentrant is
17979 not required to be thread-safe.17980 Applications wishing to check for error situations should set *errno* to 0 before calling
17981 *getpwnam()*. If *getpwnam()* returns a null pointer and *errno* is non-zero, an error occurred.17982 TSF The *getpwnam_r()* function shall update the **passwd** structure pointed to by *pwd* and store a
17983 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry
17984 from the user database with a matching *name*. Storage referenced by the structure is allocated
17985 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The
17986 maximum size needed for this buffer can be determined with the {_SC_GETPW_R_SIZE_MAX}
17987 *sysconf()* parameter. A NULL pointer shall be returned at the location pointed to by *result* on
17988 error or if the requested entry is not found.17989 **RETURN VALUE**17990 The *getpwnam()* function shall return a pointer to a **struct passwd** with the structure as defined
17991 in <pwd.h> with a matching entry if found. A null pointer shall be returned if the requested
17992 entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.17993 The return value may point to a static area which is overwritten by a subsequent call to
17994 *getpwent()*, *getpwnam()*, or *getpwuid()*.17995 TSF If successful, the *getpwnam_r()* function shall return zero; otherwise, an error number shall be
17996 returned to indicate the error.17997 **ERRORS**17998 The *getpwnam()* and *getpwnam_r()* functions may fail if:

17999 [EIO] An I/O error has occurred.

18000 [EINTR] A signal was caught during *getpwnam()*.

18001 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

18002 [ENFILE] The maximum allowable number of files is currently open in the system.

18003 The *getpwnam_r()* function may fail if:18004 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to
18005 be referenced by the resulting **passwd** structure.

18006 **EXAMPLES**18007 **Getting an Entry for the Login Name**

18008 The following example uses the *getlogin()* function to return the name of the user who logged in;
 18009 this information is passed to the *getpwnam()* function to get the user database entry for that user.

```

18010 #include <sys/types.h>
18011 #include <pwd.h>
18012 #include <unistd.h>
18013 #include <stdio.h>
18014 #include <stdlib.h>
18015 ...
18016 char *lgn;
18017 struct passwd *pw;
18018 ...
18019 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
18020     fprintf(stderr, "Get of user information failed.\n"); exit(1);
18021 }
18022 ...

```

18023 **APPLICATION USAGE**

18024 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns
 18025 the name associated with the effective user ID of the process; *getlogin()* returns the name
 18026 associated with the current login activity; and *getpwuid(getuid())* returns the name associated
 18027 with the real user ID of the process.

18028 The *getpwnam_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 18029 possibly using a static data area that may be overwritten by each call.

18030 **RATIONALE**

18031 None.

18032 **FUTURE DIRECTIONS**

18033 None.

18034 **SEE ALSO**

18035 *getpwuid()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**, **<pwd.h>**,
 18036 **<sys/types.h>**

18037 **CHANGE HISTORY**

18038 First released in Issue 1. Derived from System V Release 2.0.

18039 **Issue 5**

18040 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 18041 VALUE section.

18042 The *getpwnam_r()* function is included for alignment with the POSIX Threads Extension.

18043 A note indicating that the *getpwnam()* function need not be reentrant is added to the
 18044 DESCRIPTION.

18045 **Issue 6**

18046 The *getpwnam_r()* function is marked as part of the Thread-Safe Functions option.

18047 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION
 18048 describing matching the *name*.

- 18049 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.
- 18050 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 18051 The following new requirements on POSIX implementations derive from alignment with the
- 18052 Single UNIX Specification:
- 18053 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
 - 18054 required for conforming implementations of previous POSIX specifications, it was not
 - 18055 required for UNIX applications.
 - 18056 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
 - 18057 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.
- 18058 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
- 18059 its avoidance of possibly using a static data area.
- 18060 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
- 18061 buffer from *bufsize* characters to bytes.

18062 **NAME**

18063 getpwuid, getpwuid_r — search user database for a user ID

18064 **SYNOPSIS**

18065 #include <pwd.h>

18066 struct passwd *getpwuid(uid_t uid);

18067 TSF int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
18068 size_t bufsize, struct passwd **result);

18069

18070 **DESCRIPTION**18071 The *getpwuid()* function shall search the user database for an entry with a matching *uid*.18072 The *getpwuid()* function need not be reentrant. A function that is not required to be reentrant is
18073 not required to be thread-safe.18074 Applications wishing to check for error situations should set *errno* to 0 before calling *getpwuid()*.
18075 If *getpwuid()* returns a null pointer and *errno* is set to non-zero, an error occurred.18076 TSF The *getpwuid_r()* function shall update the **passwd** structure pointed to by *pwd* and store a
18077 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry
18078 from the user database with a matching *uid*. Storage referenced by the structure is allocated
18079 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The
18080 maximum size needed for this buffer can be determined with the {_SC_GETPW_R_SIZE_MAX}
18081 sysconf() parameter. A NULL pointer shall be returned at the location pointed to by *result* on
18082 error or if the requested entry is not found.18083 **RETURN VALUE**18084 The *getpwuid()* function shall return a pointer to a **struct passwd** with the structure as defined in
18085 <**pwd.h**> with a matching entry if found. A null pointer shall be returned if the requested entry
18086 is not found, or an error occurs. On error, *errno* shall be set to indicate the error.18087 The return value may point to a static area which is overwritten by a subsequent call to
18088 *getpwent()*, *getpwnam()*, or *getpwuid()*.18089 TSF If successful, the *getpwuid_r()* function shall return zero; otherwise, an error number shall be
18090 returned to indicate the error.18091 **ERRORS**18092 The *getpwuid()* and *getpwuid_r()* functions may fail if:

18093 [EIO] An I/O error has occurred.

18094 [EINTR] A signal was caught during *getpwuid()*.

18095 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

18096 [ENFILE] The maximum allowable number of files is currently open in the system.

18097 The *getpwuid_r()* function may fail if:18098 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to
18099 be referenced by the resulting **passwd** structure.

18100 **EXAMPLES**18101 **Getting an Entry for the Root User**

18102 The following example gets the user database entry for the user with user ID 0 (root).

```
18103 #include <sys/types.h>
18104 #include <pwd.h>
18105 ...
18106 uid_t id = 0;
18107 struct passwd *pwd;
18108 pwd = getpwuid(id);
```

18109 **Finding the Name for the Effective User ID**

18110 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to
 18111 store the structure pointer returned by the call to the *getpwuid()* function. The *geteuid()* function
 18112 shall return the effective user ID of the calling process; this is used as the search criteria for the
 18113 *getpwuid()* function. The call to *getpwuid()* shall return a pointer to the structure containing that
 18114 user ID value.

```
18115 #include <unistd.h>
18116 #include <sys/types.h>
18117 #include <pwd.h>
18118 ...
18119 struct passwd *pws;
18120 pws = getpwuid(geteuid());
```

18121 **Finding an Entry in the User Database**

18122 The following example uses *getpwuid()* to search the user database for a user ID that was
 18123 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not
 18124 found, the program prints the numeric value of the user ID for the entry.

```
18125 #include <sys/types.h>
18126 #include <pwd.h>
18127 #include <stdio.h>
18128 ...
18129 struct stat statbuf;
18130 struct passwd *pwd;
18131 ...
18132 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
18133     printf(" %-8.8s", pwd->pw_name);
18134 else
18135     printf(" %-8d", statbuf.st_uid);
```

18136 **APPLICATION USAGE**

18137 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns
 18138 the name associated with the effective user ID of the process; *getlogin()* returns the name
 18139 associated with the current login activity; and *getpwuid(getuid())* returns the name associated
 18140 with the real user ID of the process.

18141 The *getpwuid_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 18142 possibly using a static data area that may be overwritten by each call.

18143 **RATIONALE**

18144 None.

18145 **FUTURE DIRECTIONS**

18146 None.

18147 **SEE ALSO**

18148 *getpwnam()*, *geteuid()*, *getuid()*, *getlogin()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 18149 **<limits.h>**, **<pwd.h>**, **<sys/types.h>**

18150 **CHANGE HISTORY**

18151 First released in Issue 1. Derived from System V Release 2.0.

18152 **Issue 5**

18153 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 18154 VALUE section.

18155 The *getpwuid_r()* function is included for alignment with the POSIX Threads Extension.

18156 A note indicating that the *getpwuid()* function need not be reentrant is added to the
 18157 DESCRIPTION.

18158 **Issue 6**18159 The *getpwuid_r()* function is marked as part of the Thread-Safe Functions option.

18160 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION
 18161 describing matching the *uid*.

18162 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

18163 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

18164 The following new requirements on POSIX implementations derive from alignment with the
 18165 Single UNIX Specification:

18166 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
 18167 required for conforming implementations of previous POSIX specifications, it was not
 18168 required for UNIX applications.

18169 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

18170 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

18171 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 18172 its avoidance of possibly using a static data area.

18173 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
 18174 buffer from *bufsize* characters to bytes.

18175 **NAME**

18176 getrlimit, setrlimit — control maximum resource consumption

18177 **SYNOPSIS**

18178 xSI #include <sys/resource.h>

18179 int getrlimit(int resource, struct rlimit *rlp);

18180 int setrlimit(int resource, const struct rlimit *rlp);

18181

18182 **DESCRIPTION**18183 The *getrlimit()* function shall get, and the *setrlimit()* function shall set, limits on the consumption
18184 of a variety of resources.

18185 Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as
18186 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim_cur*
18187 member specifies the current or soft limit and the *rlim_max* member specifies the maximum or
18188 hard limit. Soft limits may be changed by a process to any value that is less than or equal to the
18189 hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or
18190 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both
18191 hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints
18192 described above.

18193 The value RLIM_INFINITY, defined in <sys/resource.h>, shall be considered to be larger than
18194 any other limit value. If a call to *getrlimit()* returns RLIM_INFINITY for a resource, it means the
18195 implementation shall not enforce limits on that resource. Specifying RLIM_INFINITY as any
18196 resource limit value on a successful call to *setrlimit()* shall inhibit enforcement of that resource
18197 limit.

18198 The following resources are defined:

18199 RLIMIT_CORE This is the maximum size of a **core** file, in bytes, that may be created by a
18200 process. A limit of 0 shall prevent the creation of a **core** file. If this limit is
18201 exceeded, the writing of a **core** file shall terminate at this size.

18202 RLIMIT_CPU This is the maximum amount of CPU time, in seconds, used by a process.
18203 If this limit is exceeded, SIGXCPU shall be generated for the process. If
18204 the process is catching or ignoring SIGXCPU, or all threads belonging to
18205 that process are blocking SIGXCPU, the behavior is unspecified.

18206 RLIMIT_DATA This is the maximum size of a process' data segment, in bytes. If this limit
18207 is exceeded, the *malloc()* function shall fail with *errno* set to [ENOMEM].

18208 RLIMIT_FSIZE This is the maximum size of a file, in bytes, that may be created by a
18209 process. If a write or truncate operation would cause this limit to be
18210 exceeded, SIGXFSZ shall be generated for the thread. If the thread is
18211 blocking, or the process is catching or ignoring SIGXFSZ, continued
18212 attempts to increase the size of a file from end-of-file to beyond the limit
18213 shall fail with *errno* set to [EFBIG].

18214 RLIMIT_NOFILE This is a number one greater than the maximum value that the system
18215 may assign to a newly-created descriptor. If this limit is exceeded,
18216 functions that allocate a file descriptor shall fail with *errno* set to
18217 [EMFILE]. This limit constrains the number of file descriptors that a
18218 process may allocate.

18219 RLIMIT_STACK This is the maximum size of a process' stack, in bytes. The
18220 implementation does not automatically grow the stack beyond this limit.

18221		If this limit is exceeded, SIGSEGV shall be generated for the thread. If the
18222		thread is blocking SIGSEGV, or the process is ignoring or catching
18223		SIGSEGV and has not made arrangements to use an alternate stack, the
18224		disposition of SIGSEGV shall be set to SIG_DFL before it is generated.
18225	RLIMIT_AS	This is the maximum size of a process' total available memory, in bytes. If
18226		this limit is exceeded, the <i>malloc()</i> and <i>mmap()</i> functions shall fail with
18227		<i>errno</i> set to [ENOMEM]. In addition, the automatic stack growth fails
18228		with the effects outlined above.
18229		When using the <i>getrlimit()</i> function, if a resource limit can be represented correctly in an object
18230		of type rlim_t , then its representation is returned; otherwise, if the value of the resource limit is
18231		equal to that of the corresponding saved hard limit, the value returned shall be
18232		RLIM_SAVED_MAX; otherwise, the value returned shall be RLIM_SAVED_CUR.
18233		When using the <i>setrlimit()</i> function, if the requested new limit is RLIM_INFINITY, the new limit
18234		shall be “no limit”; otherwise, if the requested new limit is RLIM_SAVED_MAX, the new limit
18235		shall be the corresponding saved hard limit; otherwise, if the requested new limit is
18236		RLIM_SAVED_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the
18237		new limit shall be the requested value. In addition, if the corresponding saved limit can be
18238		represented correctly in an object of type rlim_t then it shall be overwritten with the new limit.
18239		The result of setting a limit to RLIM_SAVED_MAX or RLIM_SAVED_CUR is unspecified unless
18240		a previous call to <i>getrlimit()</i> returned that value as the soft or hard limit for the corresponding
18241		resource limit.
18242		The determination of whether a limit can be correctly represented in an object of type rlim_t is
18243		implementation-defined. For example, some implementations permit a limit whose value is
18244		greater than RLIM_INFINITY and others do not.
18245		The <i>exec</i> family of functions shall cause resource limits to be saved.
18246	RETURN VALUE	
18247		Upon successful completion, <i>getrlimit()</i> and <i>setrlimit()</i> shall return 0. Otherwise, these functions
18248		shall return -1 and set <i>errno</i> to indicate the error.
18249	ERRORS	
18250		The <i>getrlimit()</i> and <i>setrlimit()</i> functions shall fail if:
18251	[EINVAL]	An invalid <i>resource</i> was specified; or in a <i>setrlimit()</i> call, the new <i>rlim_cur</i>
18252		exceeds the new <i>rlim_max</i> .
18253	[EPERM]	The limit specified to <i>setrlimit()</i> would have raised the maximum limit value,
18254		and the calling process does not have appropriate privileges.
18255		The <i>setrlimit()</i> function may fail if:
18256	[EINVAL]	The limit specified cannot be lowered because current usage is already higher
18257		than the limit.

18258 **EXAMPLES**

18259 None.

18260 **APPLICATION USAGE**

18261 If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the value
 18262 of {_POSIX_OPEN_MAX} from <limits.h>, unexpected behavior may occur. |

18263 If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the
 18264 highest currently open file descriptor +1, unexpected behavior may occur. |

18265 **RATIONALE**

18266 None.

18267 **FUTURE DIRECTIONS**

18268 None.

18269 **SEE ALSO**

18270 *exec*, *fork()*, *malloc()*, *open()*, *sigaltstack()*, *sysconf()*, *ulimit()*, the Base Definitions volume of
 18271 IEEE Std 1003.1-2001, <stropts.h>, <sys/resource.h>

18272 **CHANGE HISTORY**

18273 First released in Issue 4, Version 2.

18274 **Issue 5**

18275 Moved from X/OPEN UNIX extension to BASE.

18276 An APPLICATION USAGE section is added.

18277 Large File Summit extensions are added.

18278 **Issue 6**

18279 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for |
 18280 RLIMIT_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing |
 18281 with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of |
 18282 a process attempting to set the hard or soft limit for RLIMIT_NOFILE less than the highest |
 18283 currently open file descriptor +1. |

18284 **NAME**

18285 getrusage — get information about resource utilization

18286 **SYNOPSIS**

18287 xSI #include <sys/resource.h>

18288 int getrusage(int *who*, struct rusage **r_usage*);

18289

18290 **DESCRIPTION**

18291 The *getrusage()* function shall provide measures of the resources used by the current process or
 18292 its terminated and waited-for child processes. If the value of the *who* argument is
 18293 RUSAGE_SELF, information shall be returned about resources used by the current process. If the
 18294 value of the *who* argument is RUSAGE_CHILDREN, information shall be returned about
 18295 resources used by the terminated and waited-for children of the current process. If the child is
 18296 never waited for (for example, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to
 18297 SIG_IGN), the resource information for the child process is discarded and not included in the
 18298 resource information provided by *getrusage()*.

18299 The *r_usage* argument is a pointer to an object of type **struct rusage** in which the returned
 18300 information is stored.

18301 **RETURN VALUE**

18302 Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*
 18303 set to indicate the error.

18304 **ERRORS**18305 The *getrusage()* function shall fail if:

18306 [EINVAL] The value of the *who* argument is not valid.

18307 **EXAMPLES**18308 **Using getrusage()**

18309 The following example returns information about the resources used by the current process.

```
18310       #include <sys/resource.h>
18311       ...
18312       int who = RUSAGE_SELF;
18313       struct rusage usage;
18314       int ret;

18315       ret = getrusage(who, &usage);
```

18316 **APPLICATION USAGE**

18317 None.

18318 **RATIONALE**

18319 None.

18320 **FUTURE DIRECTIONS**

18321 None.

18322 **SEE ALSO**

18323 *exit()*, *sigaction()*, *time()*, *times()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 18324 <sys/resource.h>

18325 **CHANGE HISTORY**

18326 First released in Issue 4, Version 2.

18327 **Issue 5**

18328 Moved from X/OPEN UNIX extension to BASE.

18329 **NAME**

18330 gets — get a string from a stdin stream

18331 **SYNOPSIS**

18332 #include <stdio.h>

18333 char *gets(char *s);

18334 **DESCRIPTION**

18335 CX The functionality described on this reference page is aligned with the ISO C standard. Any
18336 conflict between the requirements described here and the ISO C standard is unintentional. This
18337 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18338 The *gets()* function shall read bytes from the standard input stream, *stdin*, into the array pointed
18339 to by *s*, until a <newline> is read or an end-of-file condition is encountered. Any <newline> shall
18340 be discarded and a null byte shall be placed immediately after the last byte read into the array.

18341 CX The *gets()* function may mark the *st_atime* field of the file associated with *stream* for update. The
18342 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
18343 *fread()*, *getc()*, *getchar()*, *gets()*, *fscanf()*, or *scanf()* using *stream* that returns data not supplied by
18344 a prior call to *ungetc()*.

18345 **RETURN VALUE**

18346 Upon successful completion, *gets()* shall return *s*. If the stream is at end-of-file, the end-of-file
18347 indicator for the stream shall be set and *gets()* shall return a null pointer. If a read error occurs,
18348 CX the error indicator for the stream shall be set, *gets()* shall return a null pointer, and set *errno* to
18349 indicate the error.

18350 **ERRORS**18351 Refer to *fgetc()*.18352 **EXAMPLES**

18353 None.

18354 **APPLICATION USAGE**

18355 Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of
18356 *fgets()* is recommended.

18357 Since the user cannot specify the length of the buffer passed to *gets()*, use of this function is
18358 discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in
18359 such a way as to cause applications to fail, or possible system security violations.

18360 It is recommended that the *fgets()* function should be used to read input lines.

18361 **RATIONALE**

18362 None.

18363 **FUTURE DIRECTIONS**

18364 None.

18365 **SEE ALSO**18366 *feof()*, *ferror()*, *fgets()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>18367 **CHANGE HISTORY**

18368 First released in Issue 1. Derived from Issue 1 of the SVID.

18369 **Issue 6**

18370 Extensions beyond the ISO C standard are marked.

18371 NAME

18372 getservbyname, getservbyport, getservent — network services database functions

18373 SYNOPSIS

18374 #include <netdb.h>

18375 struct servent *getservbyname(const char *name, const char *proto);

18376 struct servent *getservbyport(int port, const char *proto);

18377 struct servent *getservent(void);

18378 DESCRIPTION

18379 Refer to *endservent()*.

18380 **NAME**18381 `getsid` — get the process group ID of a session leader18382 **SYNOPSIS**18383 XSI `#include <unistd.h>`18384 `pid_t getsid(pid_t pid);`

18385

18386 **DESCRIPTION**18387 The `getsid()` function shall obtain the process group ID of the process that is the session leader of
18388 the process specified by *pid*. If *pid* is **(pid_t)0**, it specifies the calling process.18389 **RETURN VALUE**18390 Upon successful completion, `getsid()` shall return the process group ID of the session leader of
18391 the specified process. Otherwise, it shall return **(pid_t)-1** and set *errno* to indicate the error.18392 **ERRORS**18393 The `getsid()` function shall fail if:18394 [EPERM] The process specified by *pid* is not in the same session as the calling process,
18395 and the implementation does not allow access to the process group ID of the
18396 session leader of that process from the calling process.18397 [ESRCH] There is no process with a process ID equal to *pid*.18398 **EXAMPLES**

18399 None.

18400 **APPLICATION USAGE**

18401 None.

18402 **RATIONALE**

18403 None.

18404 **FUTURE DIRECTIONS**

18405 None.

18406 **SEE ALSO**18407 *exec*, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*, the Base Definitions volume of
18408 IEEE Std 1003.1-2001, **<unistd.h>**18409 **CHANGE HISTORY**

18410 First released in Issue 4, Version 2.

18411 **Issue 5**

18412 Moved from X/OPEN UNIX extension to BASE.

18413 **NAME**

18414 getsockname — get the socket name

18415 **SYNOPSIS**

18416 #include <sys/socket.h>

```
18417 int getsockname(int socket, struct sockaddr *restrict address,  
18418                 socklen_t *restrict address_len);
```

18419 **DESCRIPTION**

18420 The *getsockname()* function shall retrieve the locally-bound name of the specified socket, store
18421 this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of
18422 this address in the object pointed to by the *address_len* argument.

18423 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
18424 the stored address shall be truncated.

18425 If the socket has not been bound to a local name, the value stored in the object pointed to by
18426 *address* is unspecified.

18427 **RETURN VALUE**

18428 Upon successful completion, 0 shall be returned, the *address* argument shall point to the address
18429 of the socket, and the *address_len* argument shall point to the length of the address. Otherwise, -1
18430 shall be returned and *errno* set to indicate the error.

18431 **ERRORS**18432 The *getsockname()* function shall fail if:18433 [EBADF] The *socket* argument is not a valid file descriptor.18434 [ENOTSOCK] The *socket* argument does not refer to a socket.

18435 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

18436 The *getsockname()* function may fail if:

18437 [EINVAL] The socket has been shut down.

18438 [ENOBUFS] Insufficient resources were available in the system to complete the function.

18439 **EXAMPLES**

18440 None.

18441 **APPLICATION USAGE**

18442 None.

18443 **RATIONALE**

18444 None.

18445 **FUTURE DIRECTIONS**

18446 None.

18447 **SEE ALSO**

18448 *accept()*, *bind()*, *getpeername()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,
18449 <sys/socket.h>

18450 **CHANGE HISTORY**

18451 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18452 The **restrict** keyword is added to the *getsockname()* prototype for alignment with the
18453 ISO/IEC 9899:1999 standard.

18454 **NAME**

18455 getsockopt — get the socket options

18456 **SYNOPSIS**

18457 #include <sys/socket.h>

```
18458 int getsockopt(int socket, int level, int option_name,
18459               void *restrict option_value, socklen_t *restrict option_len);
```

18460 **DESCRIPTION**18461 The *getsockopt()* function manipulates options associated with a socket.

18462 The *getsockopt()* function shall retrieve the value for the option specified by the *option_name*
 18463 argument for the socket specified by the *socket* argument. If the size of the option value is greater
 18464 than *option_len*, the value stored in the object pointed to by the *option_value* argument shall be
 18465 silently truncated. Otherwise, the object pointed to by the *option_len* argument shall be modified
 18466 to indicate the actual length of the value.

18467 The *level* argument specifies the protocol level at which the option resides. To retrieve options at
 18468 the socket level, specify the *level* argument as SOL_SOCKET. To retrieve options at other levels,
 18469 supply the appropriate level identifier for the protocol controlling the option. For example, to
 18470 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to
 18471 IPPROTO_TCP as defined in the <netinet/in.h> header.

18472 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*
 18473 function.

18474 The *option_name* argument specifies a single option to be retrieved. It can be one of the following
 18475 values defined in <sys/socket.h>:

18476 SO_DEBUG Reports whether debugging information is being recorded. This option
 18477 shall store an **int** value. This is a Boolean option.

18478 SO_ACCEPTCONN Reports whether socket listening is enabled. This option shall store an **int**
 18479 value. This is a Boolean option.

18480 SO_BROADCAST Reports whether transmission of broadcast messages is supported, if this
 18481 is supported by the protocol. This option shall store an **int** value. This is a
 18482 Boolean option.

18483 SO_REUSEADDR Reports whether the rules used in validating addresses supplied to *bind()*
 18484 should allow reuse of local addresses, if this is supported by the protocol.
 18485 This option shall store an **int** value. This is a Boolean option.

18486 SO_KEEPALIVE Reports whether connections are kept active with periodic transmission
 18487 of messages, if this is supported by the protocol.

18488 If the connected socket fails to respond to these messages, the connection
 18489 shall be broken and threads writing to that socket shall be notified with a
 18490 SIGPIPE signal. This option shall store an **int** value. This is a Boolean
 18491 option.

18492 SO_LINGER Reports whether the socket lingers on *close()* if data is present. If
 18493 SO_LINGER is set, the system blocks the process during *close()* until it
 18494 can transmit the data or until the end of the interval indicated by the
 18495 *l_linger* member, whichever comes first. If SO_LINGER is not specified,
 18496 and *close()* is issued, the system handles the call in a way that allows the
 18497 process to continue as quickly as possible. This option shall store a **linger**
 18498 structure.

18499	SO_OOBINLINE	Reports whether the socket leaves received out-of-band data (data marked urgent) inline. This option shall store an int value. This is a Boolean option.
18500		
18501		
18502	SO_SNDBUF	Reports send buffer size information. This option shall store an int value.
18503	SO_RCVBUF	Reports receive buffer size information. This option shall store an int value.
18504		
18505	SO_ERROR	Reports information about error status and clears it. This option shall store an int value.
18506		
18507	SO_TYPE	Reports the socket type. This option shall store an int value. Socket types are described in Section 2.10.6 (on page 59).
18508		
18509	SO_DONTROUTE	Reports whether outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option shall store an int value. This is a Boolean option.
18510		
18511		
18512		
18513		
18514		
18515	SO_RCVLOWAT	Reports the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option shall store an int value. Note that not all implementations allow this option to be retrieved.
18516		
18517		
18518		
18519		
18520		
18521		
18522		
18523		
18524	SO_RCVTIMEO	Reports the timeout value for input operations. This option shall store a timeval structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data was received. The default for this option is zero, which indicates that a receive operation shall not time out. Note that not all implementations allow this option to be retrieved.
18525		
18526		
18527		
18528		
18529		
18530		
18531		
18532	SO_SNDLOWAT	Reports the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option shall store an int value. Note that not all implementations allow this option to be retrieved.
18533		
18534		
18535		
18536		
18537	SO_SNDTIMEO	Reports the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data was sent. The default for this option is zero, which indicates that a send operation shall not time out. The option shall store a timeval structure. Note that not all implementations allow this option to be retrieved.
18538		
18539		
18540		
18541		
18542		
18543		
18544		
18545		
		For Boolean options, a zero value indicates that the option is disabled and a non-zero value indicates that the option is enabled.

18546 **RETURN VALUE**

18547 Upon successful completion, *getsockopt()* shall return 0; otherwise, -1 shall be returned and *errno*
18548 set to indicate the error.

18549 **ERRORS**

18550 The *getsockopt()* function shall fail if:

18551 [EBADF] The *socket* argument is not a valid file descriptor.

18552 [EINVAL] The specified option is invalid at the specified socket level.

18553 [ENOPROTOOPT]

18554 The option is not supported by the protocol.

18555 [ENOTSOCK] The *socket* argument does not refer to a socket.

18556 The *getsockopt()* function may fail if:

18557 [EACCES] The calling process does not have the appropriate privileges.

18558 [EINVAL] The socket has been shut down.

18559 [ENOBUFS] Insufficient resources are available in the system to complete the function.

18560 **EXAMPLES**

18561 None.

18562 **APPLICATION USAGE**

18563 None.

18564 **RATIONALE**

18565 None.

18566 **FUTURE DIRECTIONS**

18567 None.

18568 **SEE ALSO**

18569 *bind()*, *close()*, *endprotoent()*, *setsockopt()*, *socket()*, the Base Definitions volume of
18570 IEEE Std 1003.1-2001, <sys/socket.h>, <netinet/in.h>

18571 **CHANGE HISTORY**

18572 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18573 The **restrict** keyword is added to the *getsockopt()* prototype for alignment with the
18574 ISO/IEC 9899:1999 standard.

18575 **NAME**

18576 getsubopt — parse suboption arguments from a string

18577 **SYNOPSIS**

18578 xSI #include <stdlib.h>

18579 int getsubopt(char **optionp, char * const *keylistp, char **valuep);

18580

18581 **DESCRIPTION**18582 The *getsubopt()* function shall parse suboption arguments in a flag argument. Such options often
18583 result from the use of *getopt()*.18584 The *getsubopt()* argument *optionp* is a pointer to a pointer to the option argument string. The
18585 suboption arguments shall be separated by commas and each may consist of either a single
18586 token, or a token-value pair separated by an equal sign.18587 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified
18588 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in
18589 **optionp*. Since commas delimit suboption arguments in *optionp*, they should not appear in any of
18590 the strings pointed to by *keylistp*. Similarly, because an equal sign separates a token from its
18591 value, the application should not include an equal sign in any of the strings pointed to by
18592 *keylistp*.18593 The *valuep* argument is the address of a value string pointer.18594 If a comma appears in *optionp*, it shall be interpreted as a suboption separator. After commas
18595 have been processed, if there are one or more equal signs in a suboption string, the first equal
18596 sign in any suboption string shall be interpreted as a separator between a token and a value.
18597 Subsequent equal signs in a suboption string shall be interpreted as part of the value.18598 If the string at **optionp* contains only one suboption argument (equivalently, no commas),
18599 *getsubopt()* shall update **optionp* to point to the null character at the end of the string. Otherwise,
18600 it shall isolate the suboption argument by replacing the comma separator with a null character,
18601 and shall update **optionp* to point to the start of the next suboption argument. If the suboption
18602 argument has an associated value (equivalently, contains an equal sign), *getsubopt()* shall update
18603 **valuep* to point to the value's first character. Otherwise, it shall set **valuep* to a null pointer. The
18604 calling application may use this information to determine whether the presence or absence of a
18605 value for the suboption is an error.18606 Additionally, when *getsubopt()* fails to match the suboption argument with a token in the *keylistp*
18607 array, the calling application should decide if this is an error, or if the unrecognized option
18608 should be processed in another way.18609 **RETURN VALUE**18610 The *getsubopt()* function shall return the index of the matched token string, or -1 if no token
18611 strings were matched.18612 **ERRORS**

18613 No errors are defined.

18614 **EXAMPLES**

```

18615     #include <stdio.h>
18616     #include <stdlib.h>

18617     int do_all;
18618     const char *type;
18619     int read_size;
18620     int write_size;
18621     int read_only;

18622     enum
18623     {
18624         RO_OPTION = 0,
18625         RW_OPTION,
18626         READ_SIZE_OPTION,
18627         WRITE_SIZE_OPTION
18628     };

18629     const char *mount_opts[] =
18630     {
18631         [RO_OPTION] = "ro",
18632         [RW_OPTION] = "rw",
18633         [READ_SIZE_OPTION] = "rsize",
18634         [WRITE_SIZE_OPTION] = "wsize",
18635         NULL
18636     };

18637     int
18638     main(int argc, char *argv[])
18639     {
18640         char *subopts, *value;
18641         int opt;

18642         while ((opt = getopt(argc, argv, "at:o:")) != -1)
18643             switch(opt)
18644             {
18645                 case 'a':
18646                     do_all = 1;
18647                     break;
18648                 case 't':
18649                     type = optarg;
18650                     break;
18651                 case 'o':
18652                     subopts = optarg;
18653                     while (*subopts != '\0')
18654                         switch(getsubopt(&subopts, mount_opts, &value))
18655                         {
18656                             case RO_OPTION:
18657                                 read_only = 1;
18658                                 break;
18659                             case RW_OPTION:
18660                                 read_only = 0;
18661                                 break;
18662                             case READ_SIZE_OPTION:

```

```

18663         if (value == NULL)
18664             abort();
18665         read_size = atoi(value);
18666         break;
18667     case WRITE_SIZE_OPTION:
18668         if (value == NULL)
18669             abort();
18670         write_size = atoi(value);
18671         break;
18672     default:
18673         /* Unknown suboption. */
18674         printf("Unknown suboption '%s'\n", value);
18675         break;
18676     }
18677     break;
18678 default:
18679     abort();
18680 }
18681
18681     /* Do the real work. */
18682     return 0;
18683 }

```

18684 Parsing Suboptions

18685 The following example uses the *getsubopt()* function to parse a *value* argument in the *optarg*
 18686 external variable returned by a call to *getopt()*.

```

18687 #include <stdlib.h>
18688 ...
18689 char *tokens[] = {"HOME", "PATH", "LOGNAME", (char *) NULL };
18690 char *value;
18691 int opt, index;
18692
18692 while ((opt = getopt(argc, argv, "e:")) != -1) {
18693     switch(opt) {
18694     case 'e' :
18695         while ((index = getsubopt(&optarg, tokens, &value)) != -1) {
18696             switch(index) {
18697             ...
18698             }
18699             break;
18700         ...
18701     }
18702 }
18703 ...

```

18704 APPLICATION USAGE

18705 None.

18706 RATIONALE

18707 None.

18708 **FUTURE DIRECTIONS**

18709 None.

18710 **SEE ALSO**18711 *getopt()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>18712 **CHANGE HISTORY**

18713 First released in Issue 4, Version 2.

18714 **Issue 5**

18715 Moved from X/OPEN UNIX extension to BASE.

18716 **Issue 6**18717 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error |
18718 in the SYNOPSIS. |

18719 **NAME**

18720 gettimeofday — get the date and time

18721 **SYNOPSIS**

18722 xSI #include <sys/time.h>

18723 int gettimeofday(struct timeval *restrict tp, void *restrict tzp);

18724

18725 **DESCRIPTION**

18726 The *gettimeofday()* function shall obtain the current time, expressed as seconds and
18727 microseconds since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The
18728 resolution of the system clock is unspecified.

18729 If *tzp* is not a null pointer, the behavior is unspecified.

18730 **RETURN VALUE**

18731 The *gettimeofday()* function shall return 0 and no value shall be reserved to indicate an error.

18732 **ERRORS**

18733 No errors are defined.

18734 **EXAMPLES**

18735 None.

18736 **APPLICATION USAGE**

18737 None.

18738 **RATIONALE**

18739 None.

18740 **FUTURE DIRECTIONS**

18741 None.

18742 **SEE ALSO**

18743 *ctime()*, *ftime()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/time.h>

18744 **CHANGE HISTORY**

18745 First released in Issue 4, Version 2.

18746 **Issue 5**

18747 Moved from X/OPEN UNIX extension to BASE.

18748 **Issue 6**

18749 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since
18750 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*
18751 functions.

18752 The **restrict** keyword is added to the *gettimeofday()* prototype for alignment with the
18753 ISO/IEC 9899:1999 standard.

18754 **NAME**

18755 getuid — get a real user ID

18756 **SYNOPSIS**

18757 #include <unistd.h>

18758 uid_t getuid(void);

18759 **DESCRIPTION**

18760 The *getuid()* function shall return the real user ID of the calling process.

18761 **RETURN VALUE**

18762 The *getuid()* function shall always be successful and no return value is reserved to indicate the error.

18764 **ERRORS**

18765 No errors are defined.

18766 **EXAMPLES**18767 **Setting the Effective User ID to the Real User ID**

18768 The following example sets the effective user ID and the real user ID of the current process to the real user ID of the caller.

```
18770           #include <unistd.h>
18771           #include <sys/types.h>
18772           ...
18773           setreuid(getuid(), getuid());
18774           ...
```

18775 **APPLICATION USAGE**

18776 None.

18777 **RATIONALE**

18778 None.

18779 **FUTURE DIRECTIONS**

18780 None.

18781 **SEE ALSO**

18782 *getegid()*, *geteuid()*, *getgid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

18784 **CHANGE HISTORY**

18785 First released in Issue 1. Derived from Issue 1 of the SVID.

18786 **Issue 6**

18787 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

18788 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 18790
 - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 18791
- 18792

18793 NAME

18794 getutxent, getutxid, getutxline — get user accounting database entries

18795 SYNOPSIS

18796 xSI #include <utmpx.h>

18797 struct utmpx *getutxent(void);

18798 struct utmpx *getutxid(const struct utmpx *id);

18799 struct utmpx *getutxline(const struct utmpx *line);

18800

18801 DESCRIPTION

18802 Refer to *endutxent()*.

18803 **NAME**

18804 getwc — get a wide character from a stream

18805 **SYNOPSIS**

18806 #include <stdio.h>

18807 #include <wchar.h>

18808 wint_t getwc(FILE **stream*);18809 **DESCRIPTION**

18810 cx The functionality described on this reference page is aligned with the ISO C standard. Any
18811 conflict between the requirements described here and the ISO C standard is unintentional. This
18812 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18813 The *getwc()* function shall be equivalent to *fgetwc()*, except that if it is implemented as a macro it
18814 may evaluate *stream* more than once, so the argument should never be an expression with side
18815 effects.

18816 **RETURN VALUE**18817 Refer to *fgetwc()*.18818 **ERRORS**18819 Refer to *fgetwc()*.18820 **EXAMPLES**

18821 None.

18822 **APPLICATION USAGE**

18823 Since it may be implemented as a macro, *getwc()* may treat incorrectly a *stream* argument with
18824 side effects. In particular, *getwc(*f++)* does not necessarily work as expected. Therefore, use of
18825 this function is not recommended; *fgetwc()* should be used instead.

18826 **RATIONALE**

18827 None.

18828 **FUTURE DIRECTIONS**

18829 None.

18830 **SEE ALSO**18831 *fgetwc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>, <wchar.h>18832 **CHANGE HISTORY**

18833 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
18834 draft.

18835 **Issue 5**

18836 The Optional Header (OH) marking is removed from <stdio.h>.

18837 NAME

18838 getwchar — get a wide character from a stdin stream

18839 SYNOPSIS

18840 #include <wchar.h>

18841 wint_t getwchar(void);

18842 DESCRIPTION

18843 cx The functionality described on this reference page is aligned with the ISO C standard. Any
18844 conflict between the requirements described here and the ISO C standard is unintentional. This
18845 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18846 The *getwchar()* function shall be equivalent to *getwc(stdin)*.

18847 RETURN VALUE

18848 Refer to *fgetwc()*.

18849 ERRORS

18850 Refer to *fgetwc()*.

18851 EXAMPLES

18852 None.

18853 APPLICATION USAGE

18854 If the **wint_t** value returned by *getwchar()* is stored into a variable of type **wchar_t** and then
18855 compared against the **wint_t** macro WEOF, the result may be incorrect. Only the **wint_t** type is
18856 guaranteed to be able to represent any wide character and WEOF.

18857 RATIONALE

18858 None.

18859 FUTURE DIRECTIONS

18860 None.

18861 SEE ALSO

18862 *fgetwc()*, *getwc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>

18863 CHANGE HISTORY

18864 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
18865 draft.

18866 NAME

18867 getwd — get the current working directory pathname (**LEGACY**)

18868 SYNOPSIS

18869 xSI #include <unistd.h>

18870 char *getwd(char *path_name);

18871

18872 DESCRIPTION

18873 The *getwd()* function shall determine an absolute pathname of the current working directory of
18874 the calling process, and copy a string containing that pathname into the array pointed to by the
18875 *path_name* argument.

18876 If the length of the pathname of the current working directory is greater than ({PATH_MAX}+1)
18877 including the null byte, *getwd()* shall fail and return a null pointer.

18878 RETURN VALUE

18879 Upon successful completion, a pointer to the string containing the absolute pathname of the
18880 current working directory shall be returned. Otherwise, *getwd()* shall return a null pointer and
18881 the contents of the array pointed to by *path_name* are undefined.

18882 ERRORS

18883 No errors are defined.

18884 EXAMPLES

18885 None.

18886 APPLICATION USAGE

18887 For applications portability, the *getcwd()* function should be used to determine the current
18888 working directory instead of *getwd()*.

18889 RATIONALE

18890 Since the user cannot specify the length of the buffer passed to *getwd()*, use of this function is
18891 discouraged. The length of a pathname described in {PATH_MAX} is file system-dependent and
18892 may vary from one mount point to another, or might even be unlimited. It is possible to
18893 overflow this buffer in such a way as to cause applications to fail, or possible system security
18894 violations.

18895 It is recommended that the *getcwd()* function should be used to determine the current working
18896 directory.

18897 FUTURE DIRECTIONS

18898 This function may be withdrawn in a future version.

18899 SEE ALSO

18900 *getcwd()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

18901 CHANGE HISTORY

18902 First released in Issue 4, Version 2.

18903 Issue 5

18904 Moved from X/OPEN UNIX extension to BASE.

18905 Issue 6

18906 This function is marked LEGACY.

18907 **NAME**

18908 glob, globfree — generate pathnames matching a pattern

18909 **SYNOPSIS**

18910 #include <glob.h>

```
18911 int glob(const char *restrict pattern, int flags,
18912         int (*errfunc)(const char *epath, int eerrno),
18913         glob_t *restrict pglob);
18914 void globfree(glob_t *pglob);
```

18915 **DESCRIPTION**

18916 The *glob()* function is a pathname generator that shall implement the rules defined in the Shell
 18917 and Utilities volume of IEEE Std 1003.1-2001, Section 2.13, Pattern Matching Notation, with
 18918 optional support for rule 3 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
 18919 2.13.3, Patterns Used for Filename Expansion.

18920 The structure type **glob_t** is defined in <glob.h> and includes at least the following members:

18921

18922

18923

18924

18925

Member Type	Member Name	Description
size_t	<i>gl_pathc</i>	Count of paths matched by <i>pattern</i> .
char **	<i>gl_pathv</i>	Pointer to a list of matched pathnames.
size_t	<i>gl_offs</i>	Slots to reserve at the beginning of <i>gl_pathv</i> .

18926 The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function
 18927 shall match all accessible pathnames against this pattern and develop a list of all pathnames that
 18928 match. In order to have access to a pathname, *glob()* requires search permission on every
 18929 component of a path except the last, and read permission on each directory of any filename
 18930 component of *pattern* that contains any of the following special characters: '*', '?', and '['.

18931 The *glob()* function shall store the number of matched pathnames into *pglob->gl_pathc* and a
 18932 pointer to a list of pointers to pathnames into *pglob->gl_pathv*. The pathnames shall be in sort
 18933 order as defined by the current setting of the *LC_COLLATE* category; see the Base Definitions
 18934 volume of IEEE Std 1003.1-2001, Section 7.3.2, *LC_COLLATE*. The first pointer after the last
 18935 pathname shall be a null pointer. If the pattern does not match any pathnames, the returned
 18936 number of matched paths is set to 0, and the contents of *pglob->gl_pathv* are implementation-
 18937 defined.

18938 It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function shall
 18939 allocate other space as needed, including the memory pointed to by *gl_pathv*. The *globfree()*
 18940 function shall free any space associated with *pglob* from a previous call to *glob()*.

18941 The *flags* argument is used to control the behavior of *glob()*. The value of *flags* is a bitwise-
 18942 inclusive OR of zero or more of the following constants, which are defined in <glob.h>:

18943	GLOBAL_APPEND	Append pathnames generated to the ones from a previous call to <i>glob()</i> .
18944	GLOBAL_DOOFFS	Make use of <i>pglob->gl_offs</i> . If this flag is set, <i>pglob->gl_offs</i> is used to specify how many null pointers to add to the beginning of <i>pglob->gl_pathv</i> . In other words, <i>pglob->gl_pathv</i> shall point to <i>pglob->gl_offs</i> null pointers, followed by <i>pglob->gl_pathc</i> pathname pointers, followed by a null pointer.
18949	GLOBAL_ERR	Cause <i>glob()</i> to return when it encounters a directory that it cannot open or read. Ordinarily, <i>glob()</i> continues to find matches.

18950

18951	GLOB_MARK	Each pathname that is a directory that matches <i>pattern</i> shall have a slash appended.
18952		
18953	GLOB_NOCHECK	Supports rule 3 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13.3, Patterns Used for Filename Expansion. If <i>pattern</i> does not match any pathname, then <i>glob()</i> shall return a list consisting of only <i>pattern</i> , and the number of matched pathnames is 1.
18954		
18955		
18956		
18957	GLOB_NOESCAPE	Disable backslash escaping.
18958	GLOB_NOSORT	Ordinarily, <i>glob()</i> sorts the matching pathnames according to the current setting of the <i>LC_COLLATE</i> category; see the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.2, <i>LC_COLLATE</i> . When this flag is used, the order of pathnames returned is unspecified.
18959		
18960		
18961		
18962	The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a previous call to <i>glob()</i> . The following rules apply to applications when two or more calls to <i>glob()</i> are made with the same value of <i>pglob</i> and without intervening calls to <i>globfree()</i> :	
18963		
18964		
18965	1.	The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.
18966	2.	All the calls shall set GLOB_DOOFFS, or all shall not set it.
18967	3.	After the second call, <i>pglob->gl_pathv</i> points to a list containing the following:
18968	a.	Zero or more null pointers, as specified by GLOB_DOOFFS and <i>pglob->gl_offs</i> .
18969	b.	Pointers to the pathnames that were in the <i>pglob->gl_pathv</i> list before the call, in the same order as before.
18970		
18971	c.	Pointers to the new pathnames generated by the second call, in the specified order.
18972	4.	The count returned in <i>pglob->gl_pathc</i> shall be the total number of pathnames from the two calls.
18973		
18974	5.	The application can change any of the fields after a call to <i>glob()</i> . If it does, the application shall reset them to the original value before a subsequent call, using the same <i>pglob</i> value, to <i>globfree()</i> or <i>glob()</i> with the GLOB_APPEND flag.
18975		
18976		
18977	If, during the search, a directory is encountered that cannot be opened or read and <i>errfunc</i> is not a null pointer, <i>glob()</i> calls (<i>*errfunc()</i>) with two arguments:	
18978		
18979	1.	The <i>epath</i> argument is a pointer to the path that failed.
18980	2.	The <i>errno</i> argument is the value of <i>errno</i> from the failure, as set by <i>opendir()</i> , <i>readdir()</i> , or <i>stat()</i> . (Other values may be used to report other errors not explicitly documented for those functions.)
18981		
18982		
18983	If (<i>*errfunc()</i>) is called and returns non-zero, or if the GLOB_ERR flag is set in <i>flags</i> , <i>glob()</i> shall stop the scan and return GLOB_ABORTED after setting <i>gl_pathc</i> and <i>gl_pathv</i> in <i>pglob</i> to reflect the paths already scanned. If GLOB_ERR is not set and either <i>errfunc</i> is a null pointer or (<i>*errfunc()</i>) returns 0, the error shall be ignored.	
18984		
18985		
18986		
18987	The <i>glob()</i> function shall not fail because of large files.	
18988	RETURN VALUE	
18989	Upon successful completion, <i>glob()</i> shall return 0. The argument <i>pglob->gl_pathc</i> shall return the number of matched pathnames and the argument <i>pglob->gl_pathv</i> shall contain a pointer to a null-terminated list of matched and sorted pathnames. However, if <i>pglob->gl_pathc</i> is 0, the content of <i>pglob->gl_pathv</i> is undefined.	
18990		
18991		
18992		

18993 The *globfree()* function shall not return a value.

18994 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in
18995 **<glob.h>**. The arguments *pglob->gl_pathc* and *pglob->gl_pathv* are still set as defined above.

18996 **ERRORS**

18997 The *glob()* function shall fail and return the corresponding value if:

18998 GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (**errfunc()*)
18999 returned non-zero.

19000 GLOB_NOMATCH The pattern does not match any existing pathname, and
19001 GLOB_NOCHECK was not set in flags.

19002 GLOB_NOSPACE An attempt to allocate memory failed.

19003 **EXAMPLES**

19004 One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with
19005 *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the
19006 equivalent of:

```
19007 ls -l *.c
```

19008 but for some reason:

```
19009 system("ls -l *.c")
```

19010 is not acceptable. The application could obtain approximately the same result using the
19011 sequence:

```
19012 globbuf.gl_offs = 2;  
19013 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);  
19014 globbuf.gl_pathv[0] = "ls";  
19015 globbuf.gl_pathv[1] = "-l";  
19016 execvp("ls", &globbuf.gl_pathv[0]);
```

19017 Using the same example:

```
19018 ls -l *.c *.h
```

19019 could be approximately simulated using GLOB_APPEND as follows:

```
19020 globbuf.gl_offs = 2;  
19021 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);  
19022 glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);  
19023 ...
```

19024 **APPLICATION USAGE**

19025 This function is not provided for the purpose of enabling utilities to perform pathname
19026 expansion on their arguments, as this operation is performed by the shell, and utilities are
19027 explicitly not expected to redo this. Instead, it is provided for applications that need to do
19028 pathname expansion on strings obtained from other sources, such as a pattern typed by a user or
19029 read from a file.

19030 If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

19031 Note that *gl_pathc* and *gl_pathv* have meaning even if *glob()* fails. This allows *glob()* to report
19032 partial results in the event of an error. However, if *gl_pathc* is 0, *gl_pathv* is unspecified even if
19033 *glob()* did not return an error.

19034 The GLOB_NOCHECK option could be used when an application wants to expand a pathname
19035 if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility

19036 might use this for option-arguments, for example.

19037 The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted
 19038 together with the previous pathnames. This mirrors the way that the shell handles pathname
 19039 expansion when multiple expansions are done on a command line.

19040 Applications that need tilde and parameter expansion should use *wordexp()*.

19041 RATIONALE

19042 It was claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
19043 new = (char **)malloc((n + pglob->gl_pathc + 1)
19044                      * sizeof(char *));
19045 (void) memcpy(new+n, pglob->gl_pathv,
19046              pglob->gl_pathc * sizeof(char *));
19047 (void) memset(new, 0, n * sizeof(char *));
19048 free(pglob->gl_pathv);
19049 pglob->gl_pathv = new;
```

19050 However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately
 19051 created using *malloc()*. This is not necessarily the case. An application should make no
 19052 assumptions about how the memory referenced by fields in *pglob* was allocated. It might have
 19053 been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have
 19054 been created using a different memory allocator. It is not the intent of the standard developers to
 19055 specify or imply how the memory used by *glob()* is managed.

19056 The GLOB_APPEND flag would be used when an application wants to expand several different
 19057 patterns into a single list.

19058 FUTURE DIRECTIONS

19059 None.

19060 SEE ALSO

19061 *exec*, *fnmatch()*, *opendir()*, *readdir()*, *stat()*, *wordexp()*, the Base Definitions volume of
 19062 IEEE Std 1003.1-2001, <**glob.h**>, the Shell and Utilities volume of IEEE Std 1003.1-2001

19063 CHANGE HISTORY

19064 First released in Issue 4. Derived from the ISO POSIX-2 standard.

19065 Issue 5

19066 Moved from POSIX2 C-language Binding to BASE.

19067 Issue 6

19068 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

19069 The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999
 19070 standard.

19071 **NAME**

19072 gmtime, gmtime_r — convert a time value to a broken-down UTC time

19073 **SYNOPSIS**

19074 #include <time.h>

19075 struct tm *gmtime(const time_t *timer);

19076 TSF struct tm *gmtime_r(const time_t *restrict timer,

19077 struct tm *restrict result);

19078

19079 **DESCRIPTION**

19080 CX For *gmtime()*: The functionality described on this reference page is aligned with the ISO C
 19081 standard. Any conflict between the requirements described here and the ISO C standard is
 19082 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19083 The *gmtime()* function shall convert the time in seconds since the Epoch pointed to by *timer* into
 19084 a broken-down time, expressed as Coordinated Universal Time (UTC).

19085 CX The relationship between a time in seconds since the Epoch used as an argument to *gmtime()*
 19086 and the **tm** structure (defined in the <time.h> header) is that the result shall be as specified in the
 19087 expression given in the definition of seconds since the Epoch (see the Base Definitions volume of
 19088 IEEE Std 1003.1-2001, Section 4.14, Seconds Since the Epoch), where the names in the structure
 19089 and in the expression correspond.

19090 TSF The same relationship shall apply for *gmtime_r()*.

19091 CX The *gmtime()* function need not be reentrant. A function that is not required to be reentrant is not
 19092 required to be thread-safe.

19093 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 19094 objects: a broken-down time structure and an array of type **char**. Execution of any of the
 19095 functions may overwrite the information returned in either of these objects by any of the other
 19096 functions.

19097 TSF The *gmtime_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*
 19098 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down
 19099 time is stored in the structure referred to by *result*. The *gmtime_r()* function shall also return the
 19100 address of the same structure.

19101 **RETURN VALUE**

19102 Upon successful completion, the *gmtime()* function shall return a pointer to a **struct tm**. If an
 19103 CX error is detected, *gmtime()* shall return a null pointer and set *errno* to indicate the error.

19104 TSF Upon successful completion, *gmtime_r()* shall return the address of the structure pointed to by
 19105 the argument *result*. If an error is detected, *gmtime_r()* shall return a null pointer.

19106 **ERRORS**19107 The *gmtime()* function shall fail if:

19108 CX [EOverflow] The result cannot be represented.

19109 **EXAMPLES**

19110 None.

19111 **APPLICATION USAGE**

19112 The *gmtime_r()* function is thread-safe and returns values in a user-supplied buffer instead of
19113 possibly using a static data area that may be overwritten by each call.

19114 **RATIONALE**

19115 None.

19116 **FUTURE DIRECTIONS**

19117 None.

19118 **SEE ALSO**

19119 *asctime()*, *clock()*, *ctime()*, *difftime()*, *localtime()*, *mktime()*, *strptime()*, *strptime()*, *time()*, *utime()*,
19120 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

19121 **CHANGE HISTORY**

19122 First released in Issue 1. Derived from Issue 1 of the SVID.

19123 **Issue 5**

19124 A note indicating that the *gmtime()* function need not be reentrant is added to the
19125 DESCRIPTION.

19126 The *gmtime_r()* function is included for alignment with the POSIX Threads Extension.19127 **Issue 6**19128 The *gmtime_r()* function is marked as part of the Thread-Safe Functions option.

19129 Extensions beyond the ISO C standard are marked.

19130 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
19131 its avoidance of possibly using a static data area.

19132 The **restrict** keyword is added to the *gmtime_r()* prototype for alignment with the
19133 ISO/IEC 9899:1999 standard.

19134 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [EOVERFLOW]
19135 error.

19136 **NAME**

19137 grantpt — grant access to the slave pseudo-terminal device

19138 **SYNOPSIS**

19139 xSI #include <stdlib.h>

19140 int grantpt(int *fildes*);

19141

19142 **DESCRIPTION**

19143 The *grantpt()* function shall change the mode and ownership of the slave pseudo-terminal
 19144 device associated with its master pseudo-terminal counterpart. The *fildes* argument is a file
 19145 descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to
 19146 the real UID of the calling process and the group ID shall be set to an unspecified group ID. The
 19147 permission mode of the slave pseudo-terminal shall be set to readable and writable by the
 19148 owner, and writable by the group.

19149 The behavior of the *grantpt()* function is unspecified if the application has installed a signal
 19150 handler to catch SIGCHLD signals.

19151 **RETURN VALUE**

19152 Upon successful completion, *grantpt()* shall return 0; otherwise, it shall return -1 and set *errno* to
 19153 indicate the error.

19154 **ERRORS**19155 The *grantpt()* function may fail if:

19156	[EBADF]	The <i>fildes</i> argument is not a valid open file descriptor.
19157	[EINVAL]	The <i>fildes</i> argument is not associated with a master pseudo-terminal device.
19158	[EACCES]	The corresponding slave pseudo-terminal device could not be accessed.

19159 **EXAMPLES**

19160 None.

19161 **APPLICATION USAGE**

19162 None.

19163 **RATIONALE**

19164 None.

19165 **FUTURE DIRECTIONS**

19166 None.

19167 **SEE ALSO**19168 *open()*, *ptsname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>19169 **CHANGE HISTORY**

19170 First released in Issue 4, Version 2.

19171 **Issue 5**

19172 Moved from X/OPEN UNIX extension to BASE.

19173 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

19174 NAME

19175 h_errno — error return value for network database operations

19176 SYNOPSIS

19177 OB #include <netdb.h>

19178

19179 DESCRIPTION

19180 This method of returning errors is used only in connection with obsolescent functions.

19181 The <netdb.h> header provides a declaration of *h_errno* as a modifiable lvalue of type **int**.

19182 It is unspecified whether *h_errno* is a macro or an identifier declared with external linkage. If a
19183 macro definition is suppressed in order to access an actual object, or a program defines an
19184 identifier with the name *h_errno*, the behavior is undefined.

19185 RETURN VALUE

19186 None.

19187 ERRORS

19188 No errors are defined.

19189 EXAMPLES

19190 None.

19191 APPLICATION USAGE

19192 Applications should obtain the definition of *h_errno* by the inclusion of the <netdb.h> header.

19193 RATIONALE

19194 None.

19195 FUTURE DIRECTIONS

19196 *h_errno* may be withdrawn in a future version.

19197 SEE ALSO

19198 *endhostent()*, *errno*, the Base Definitions volume of IEEE Std 1003.1-2001, <netdb.h>

19199 CHANGE HISTORY

19200 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19201 **NAME**

19202 hcreate, hdestroy, hsearch — manage hash search table

19203 **SYNOPSIS**

```
19204 xSI      #include <search.h>
19205
19206      int hcreate(size_t nel);
19207      void hdestroy(void);
19208      ENTRY *hsearch(ENTRY item, ACTION action);
```

19209 **DESCRIPTION**19210 The *hcreate()*, *hdestroy()*, and *hsearch()* functions shall manage hash search tables.

19211 The *hcreate()* function shall allocate sufficient space for the table, and the application shall ensure it is called before *hsearch()* is used. The *nel* argument is an estimate of the maximum number of entries that the table shall contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

19215 The *hdestroy()* function shall dispose of the search table, and may be followed by another call to *hcreate()*. After the call to *hdestroy()*, the data can no longer be considered accessible.

19217 The *hsearch()* function is a hash-table search routine. It shall return a pointer into a hash table indicating the location at which an entry can be found. The *item* argument is a structure of type **ENTRY** (defined in the *<search.h>* header) containing two pointers: *item.key* points to the comparison key (a **char ***), and *item.data* (a **void ***) points to any other data to be associated with that key. The comparison function used by *hsearch()* is *strcmp()*. The *action* argument is a member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

19226 These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

19228 **RETURN VALUE**

19229 The *hcreate()* function shall return 0 if it cannot allocate sufficient space for the table; otherwise, it shall return non-zero.

19231 The *hdestroy()* function shall not return a value.

19232 The *hsearch()* function shall return a null pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

19234 **ERRORS**19235 The *hcreate()* and *hsearch()* functions may fail if:

19236 [ENOMEM] Insufficient storage space is available.

19237 **EXAMPLES**

19238 The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```
19241      #include <stdio.h>
19242      #include <search.h>
19243      #include <string.h>
19244
19245      struct info {          /* This is the info stored in the table */
19246          int age, room;     /* other than the key. */
19247      };
```

```

19246     };
19247     #define NUM_EMPL    5000    /* # of elements in search table. */
19248     int main(void)
19249     {
19250         char string_space[NUM_EMPL*20];    /* Space to store strings. */
19251         struct info info_space[NUM_EMPL]; /* Space to store employee info. */
19252         char *str_ptr = string_space;      /* Next space in string_space. */
19253         struct info *info_ptr = info_space;
19254                                         /* Next space in info_space. */
19255         ENTRY item;
19256         ENTRY *found_item; /* Name to look for in table. */
19257         char name_to_find[30];
19258         int i = 0;
19259         /* Create table; no error checking is performed. */
19260         (void) hcreate(NUM_EMPL);
19261         while (scanf("%s%d%d", str_ptr, &info_ptr->age,
19262                     &info_ptr->room) != EOF && i++ < NUM_EMPL) {
19263             /* Put information in structure, and structure in item. */
19264             item.key = str_ptr;
19265             item.data = info_ptr;
19266             str_ptr += strlen(str_ptr) + 1;
19267             info_ptr++;
19268             /* Put item into table. */
19269             (void) hsearch(item, ENTER);
19270         }
19271         /* Access table. */
19272         item.key = name_to_find;
19273         while (scanf("%s", item.key) != EOF) {
19274             if ((found_item = hsearch(item, FIND)) != NULL) {
19275                 /* If item is in the table. */
19276                 (void)printf("found %s, age = %d, room = %d\n",
19277                             found_item->key,
19278                             ((struct info *)found_item->data)->age,
19279                             ((struct info *)found_item->data)->room);
19280             } else
19281                 (void)printf("no such employee %s\n", name_to_find);
19282         }
19283         return 0;
19284     }

```

19285 APPLICATION USAGE

19286 The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

19287 RATIONALE

19288 None.

19289 FUTURE DIRECTIONS

19290 None.

19291 SEE ALSO

19292 *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tsearch()*, the Base Definitions volume of
19293 IEEE Std 1003.1-2001, <**search.h**>

19294 CHANGE HISTORY

19295 First released in Issue 1. Derived from Issue 1 of the SVID.

19296 Issue 6

19297 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

19298 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

19299 **NAME**

19300 htonl, htons, ntohl, ntohs — convert values between host and network byte order

19301 **SYNOPSIS**

19302 #include <arpa/inet.h>

19303 uint32_t htonl(uint32_t *hostlong*);
19304 uint16_t htons(uint16_t *hostshort*);
19305 uint32_t ntohl(uint32_t *netlong*);
19306 uint16_t ntohs(uint16_t *netshort*);

19307 **DESCRIPTION**

19308 These functions shall convert 16-bit and 32-bit quantities between network byte order and host
19309 byte order.

19310 On some implementations, these functions are defined as macros.

19311 The **uint32_t** and **uint16_t** types are defined in <inttypes.h>.

19312 **RETURN VALUE**

19313 The *htonl()* and *htons()* functions shall return the argument value converted from host to
19314 network byte order.

19315 The *ntohl()* and *ntohs()* functions shall return the argument value converted from network to
19316 host byte order.

19317 **ERRORS**

19318 No errors are defined.

19319 **EXAMPLES**

19320 None.

19321 **APPLICATION USAGE**

19322 These functions are most often used in conjunction with IPv4 addresses and ports as returned by
19323 *gethostent()* and *getservent()*.

19324 **RATIONALE**

19325 None.

19326 **FUTURE DIRECTIONS**

19327 None.

19328 **SEE ALSO**

19329 *endhostent()*, *endservent()*, the Base Definitions volume of IEEE Std 1003.1-2001, <inttypes.h>,
19330 <arpa/inet.h>

19331 **CHANGE HISTORY**

19332 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19333 **NAME**

19334 hypot, hypotf, hypotl — Euclidean distance function

19335 **SYNOPSIS**

19336 #include <math.h>

19337 double hypot(double x, double y);

19338 float hypotf(float x, float y);

19339 long double hypotl(long double x, long double y);

19340 **DESCRIPTION**

19341 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19342 conflict between the requirements described here and the ISO C standard is unintentional. This
 19343 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19344 These functions shall compute the value of the square root of x^2+y^2 without undue overflow or
 19345 underflow.

19346 An application wishing to check for error situations should set *errno* to zero and call
 19347 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 19348 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 19349 zero, an error has occurred.

19350 **RETURN VALUE**

19351 Upon successful completion, these functions shall return the length of the hypotenuse of a
 19352 right-angled triangle with sides of length *x* and *y*.

19353 If the correct value would cause overflow, a range error shall occur and *hypot()*, *hypotf()*, and
 19354 *hypotl()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 19355 respectively.

19356 MX If *x* or *y* is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned (even if one of *x* or *y* is NaN).

19357 If *x* or *y* is NaN, and the other is not $\pm\text{Inf}$, a NaN shall be returned.

19358 If both arguments are subnormal and the correct result is subnormal, a range error may occur
 19359 and the correct result is returned.

19360 **ERRORS**

19361 These functions shall fail if:

19362 Range Error The result overflows.

19363 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 19364 then *errno* shall be set to [ERANGE]. If the integer expression
 19365 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 19366 floating-point exception shall be raised.

19367 These functions may fail if:

19368 MX Range Error The result underflows.

19369 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 19370 then *errno* shall be set to [ERANGE]. If the integer expression
 19371 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 19372 floating-point exception shall be raised.

19373 **EXAMPLES**

19374 None.

19375 **APPLICATION USAGE**19376 *hypot(x,y)*, *hypot(y,x)*, and *hypot(x, -y)* are equivalent.19377 *hypot(x, ±0)* is equivalent to *fabs(x)*.19378 Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also
19379 subnormal.19380 These functions take precautions against overflow during intermediate steps of the
19381 computation.19382 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
19383 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.19384 **RATIONALE**

19385 None.

19386 **FUTURE DIRECTIONS**

19387 None.

19388 **SEE ALSO**19389 *feclearexcept()*, *fetestexcept()*, *isnan()*, *sqrt()*, the Base Definitions volume of IEEE Std 1003.1-2001,
19390 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>19391 **CHANGE HISTORY**

19392 First released in Issue 1. Derived from Issue 1 of the SVID.

19393 **Issue 5**19394 The DESCRIPTION is updated to indicate how an application should check for an error. This
19395 text was previously published in the APPLICATION USAGE section.19396 **Issue 6**19397 The *hypot()* function is no longer marked as an extension.19398 The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999
19399 standard.19400 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
19401 revised to align with the ISO/IEC 9899:1999 standard.19402 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
19403 marked.

19404 **NAME**

19405 iconv — codeset conversion function

19406 **SYNOPSIS**19407 xSI `#include <iconv.h>`

```
19408     size_t iconv(iconv_t cd, char **restrict inbuf,
19409                 size_t *restrict inbytesleft, char **restrict outbuf,
19410                 size_t *restrict outbytesleft);
```

19411

19412 **DESCRIPTION**

19413 The *iconv()* function shall convert the sequence of characters from one codeset, in the array
 19414 specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array
 19415 specified by *outbuf*. The codesets are those specified in the *iconv_open()* call that returned the
 19416 conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first
 19417 character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer
 19418 to be converted. The *outbuf* argument points to a variable that points to the first available byte in
 19419 the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the
 19420 buffer.

19421 For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by
 19422 a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv()* is
 19423 called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft*
 19424 points to a positive value, *iconv()* shall place, into the output buffer, the byte sequence to change
 19425 the output buffer to its initial shift state. If the output buffer is not large enough to hold the
 19426 entire reset sequence, *iconv()* shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as
 19427 other than a null pointer or a pointer to a null pointer cause the conversion to take place from
 19428 the current state of the conversion descriptor.

19429 If a sequence of input bytes does not form a valid character in the specified codeset, conversion
 19430 shall stop after the previous successfully converted character. If the input buffer ends with an
 19431 incomplete character or shift sequence, conversion shall stop after the previous successfully
 19432 converted bytes. If the output buffer is not large enough to hold the entire converted input,
 19433 conversion shall stop just prior to the input bytes that would cause the output buffer to
 19434 overflow. The variable pointed to by *inbuf* shall be updated to point to the byte following the last
 19435 byte successfully used in the conversion. The value pointed to by *inbytesleft* shall be
 19436 decremented to reflect the number of bytes still not converted in the input buffer. The variable
 19437 pointed to by *outbuf* shall be updated to point to the byte following the last byte of converted
 19438 output data. The value pointed to by *outbytesleft* shall be decremented to reflect the number of
 19439 bytes still available in the output buffer. For state-dependent encodings, the conversion
 19440 descriptor shall be updated to reflect the shift state in effect at the end of the last successfully
 19441 converted byte sequence.

19442 If *iconv()* encounters a character in the input buffer that is valid, but for which an identical
 19443 character does not exist in the target codeset, *iconv()* shall perform an implementation-defined
 19444 conversion on this character.

19445 **RETURN VALUE**

19446 The *iconv()* function shall update the variables pointed to by the arguments to reflect the extent
 19447 of the conversion and return the number of non-identical conversions performed. If the entire
 19448 string in the input buffer is converted, the value pointed to by *inbytesleft* shall be 0. If the input
 19449 conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft*
 19450 shall be non-zero and *errno* shall be set to indicate the condition. If an error occurs, *iconv()* shall
 19451 return (**size_t**)-1 and set *errno* to indicate the error.

19452 **ERRORS**19453 The *iconv()* function shall fail if:

19454 [EILSEQ] Input conversion stopped due to an input byte that does not belong to the
 19455 input codeset.

19456 [E2BIG] Input conversion stopped due to lack of space in the output buffer.

19457 [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at
 19458 the end of the input buffer.

19459 The *iconv()* function may fail if:

19460 [EBADF] The *cd* argument is not a valid open conversion descriptor.

19461 **EXAMPLES**

19462 None.

19463 **APPLICATION USAGE**

19464 The *inbuf* argument indirectly points to the memory area which contains the conversion input
 19465 data. The *outbuf* argument indirectly points to the memory area which is to contain the result of
 19466 the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to
 19467 containing data that is directly representable in the ISO C standard language **char** data type. The
 19468 type of *inbuf* and *outbuf*, **char ****, does not imply that the objects pointed to are interpreted as
 19469 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that
 19470 represents a character in a given character set encoding scheme is done internally within the
 19471 codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can
 19472 contain all zero octets that are not interpreted as string terminators but as coded character data
 19473 according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and
 19474 so on) read or stored in the objects is not specified, but may be inferred for both the input and
 19475 output data by the converters determined by the *fromcode* and *toencode* arguments of *iconv_open()*.

19476 Regardless of the data type inferred by the converter, the size of the remaining space in both
 19477 input and output objects (the *inbytesleft* and *outbytesleft* arguments) is always measured in bytes.

19478 For implementations that support the conversion of state-dependent encodings, the conversion
 19479 descriptor must be able to accurately reflect the shift-state in effect at the end of the last
 19480 successful conversion. It is not required that the conversion descriptor itself be updated, which
 19481 would require it to be a pointer type. Thus, implementations are free to implement the
 19482 descriptor as a handle (other than a pointer type) by which the conversion information can be
 19483 accessed and updated.

19484 **RATIONALE**

19485 None.

19486 **FUTURE DIRECTIONS**

19487 None.

19488 **SEE ALSO**19489 *iconv_open()*, *iconv_close()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**iconv.h**>19490 **CHANGE HISTORY**

19491 First released in Issue 4. Derived from the HP-UX Manual.

19492 **Issue 6**19493 The SYNOPSIS has been corrected to align with the <**iconv.h**> reference page.

19494 The **restrict** keyword is added to the *iconv()* prototype for alignment with the
 19495 ISO/IEC 9899:1999 standard.

19496 **NAME**

19497 iconv_close — codeset conversion deallocation function

19498 **SYNOPSIS**

19499 xSI #include <iconv.h>

19500 int iconv_close(iconv_t *cd*);

19501

19502 **DESCRIPTION**19503 The *iconv_close()* function shall deallocate the conversion descriptor *cd* and all other associated
19504 resources allocated by *iconv_open()*.19505 If a file descriptor is used to implement the type **iconv_t**, that file descriptor shall be closed.19506 **RETURN VALUE**19507 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
19508 indicate the error.19509 **ERRORS**19510 The *iconv_close()* function may fail if:

19511 [EBADF] The conversion descriptor is invalid.

19512 **EXAMPLES**

19513 None.

19514 **APPLICATION USAGE**

19515 None.

19516 **RATIONALE**

19517 None.

19518 **FUTURE DIRECTIONS**

19519 None.

19520 **SEE ALSO**19521 *iconv()*, *iconv_open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**iconv.h**>19522 **CHANGE HISTORY**

19523 First released in Issue 4. Derived from the HP-UX Manual.

19524 **NAME**

19525 iconv_open — codeset conversion allocation function

19526 **SYNOPSIS**19527 xSI `#include <iconv.h>`19528 `iconv_t iconv_open(const char *tocode, const char *fromcode);`

19529

19530 **DESCRIPTION**

19531 The *iconv_open()* function shall return a conversion descriptor that describes a conversion from
 19532 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified
 19533 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion
 19534 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with
 19535 *iconv()*.

19536 Settings of *fromcode* and *tocode* and their permitted combinations are implementation-defined.

19537 A conversion descriptor shall remain valid until it is closed by *iconv_close()* or an implicit close.

19538 If a file descriptor is used to implement conversion descriptors, the FD_CLOEXEC flag shall be
 19539 set; see <fcntl.h>.

19540 **RETURN VALUE**

19541 Upon successful completion, *iconv_open()* shall return a conversion descriptor for use on
 19542 subsequent calls to *iconv()*. Otherwise, *iconv_open()* shall return (**iconv_t**)-1 and set *errno* to
 19543 indicate the error.

19544 **ERRORS**

19545 The *iconv_open()* function may fail if:

19546 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

19547 [ENFILE] Too many files are currently open in the system.

19548 [ENOMEM] Insufficient storage space is available.

19549 [EINVAL] The conversion specified by *fromcode* and *tocode* is not supported by the
 19550 implementation.

19551 **EXAMPLES**

19552 None.

19553 **APPLICATION USAGE**

19554 Some implementations of *iconv_open()* use *malloc()* to allocate space for internal buffer areas.
 19555 The *iconv_open()* function may fail if there is insufficient storage space to accommodate these
 19556 buffers.

19557 Conforming applications must assume that conversion descriptors are not valid after a call to
 19558 one of the *exec* functions.

19559 Application developers should consult the system documentation to determine the supported
 19560 codesets and their naming schemes.

19561 **RATIONALE**

19562 None.

19563 **FUTURE DIRECTIONS**

19564 None.

19565 **SEE ALSO**

19566 *iconv()*, *iconv_close()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**, **<iconv.h>**

19567 **CHANGE HISTORY**

19568 First released in Issue 4. Derived from the HP-UX Manual.

19569 NAME

19570 if_freenameindex — free memory allocated by if_nameindex

19571 SYNOPSIS

19572 #include <net/if.h>

19573 void if_freenameindex(struct if_nameindex *ptr);

19574 DESCRIPTION

19575 The *if_freenameindex()* function shall free the memory allocated by *if_nameindex()*. The *ptr*
19576 argument shall be a pointer that was returned by *if_nameindex()*. After *if_freenameindex()* has
19577 been called, the application shall not use the array of which *ptr* is the address.

19578 RETURN VALUE

19579 None.

19580 ERRORS

19581 No errors are defined.

19582 EXAMPLES

19583 None.

19584 APPLICATION USAGE

19585 None.

19586 RATIONALE

19587 None.

19588 FUTURE DIRECTIONS

19589 None.

19590 SEE ALSO

19591 *getsockopt()*, *if_indextoname()*, *if_nameindex()*, *if_nametoindex()*, *setsockopt()*, the Base Definitions
19592 volume of IEEE Std 1003.1-2001, <**net/if.h**>

19593 CHANGE HISTORY

19594 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19595 **NAME**

19596 if_indextoname — map a network interface index to its corresponding name

19597 **SYNOPSIS**

19598 #include <net/if.h>

19599 char *if_indextoname(unsigned *ifindex*, char **ifname*);

19600 **DESCRIPTION**

19601 The *if_indextoname()* function shall map an interface index to its corresponding name.

19602 When this function is called, *ifname* shall point to a buffer of at least {IF_NAMESIZE} bytes. The
19603 function shall place in this buffer the name of the interface with index *ifindex*.

19604 **RETURN VALUE**

19605 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which
19606 points to a buffer now containing the interface name. Otherwise, the function shall return a
19607 NULL pointer and set *errno* to indicate the error.

19608 **ERRORS**

19609 The *if_indextoname()* function shall fail if:

19610 [ENXIO] The interface does not exist.

19611 **EXAMPLES**

19612 None.

19613 **APPLICATION USAGE**

19614 None.

19615 **RATIONALE**

19616 None.

19617 **FUTURE DIRECTIONS**

19618 None.

19619 **SEE ALSO**

19620 *getsockopt()*, *if_freenameindex()*, *if_nameindex()*, *if_nametoindex()*, *setsockopt()*, the Base
19621 Definitions volume of IEEE Std 1003.1-2001, <net/if.h>

19622 **CHANGE HISTORY**

19623 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19624 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to
19625 {IF_NAMESIZ} in the DESCRIPTION.

19626 **NAME**

19627 if_nameindex — return all network interface names and indexes

19628 **SYNOPSIS**

19629 #include <net/if.h>

19630 struct if_nameindex *if_nameindex(void);

19631 **DESCRIPTION**

19632 The *if_nameindex()* function shall return an array of *if_nameindex* structures, one structure per
19633 interface. The end of the array is indicated by a structure with an *if_index* field of zero and an
19634 *if_name* field of NULL.

19635 Applications should call *if_freenameindex()* to release the memory that may be dynamically
19636 allocated by this function, after they have finished using it.

19637 **RETURN VALUE**

19638 An array of structures identifying local interfaces. A NULL pointer is returned upon an error,
19639 with *errno* set to indicate the error.

19640 **ERRORS**

19641 The *if_nameindex()* function may fail if:

19642 [ENOBUFS] Insufficient resources are available to complete the function.

19643 **EXAMPLES**

19644 None.

19645 **APPLICATION USAGE**

19646 None.

19647 **RATIONALE**

19648 None.

19649 **FUTURE DIRECTIONS**

19650 None.

19651 **SEE ALSO**

19652 *getsockopt()*, *if_freenameindex()*, *if_indextoname()*, *if_nametoindex()*, *setsockopt()*, the Base
19653 Definitions volume of IEEE Std 1003.1-2001, <net/if.h>

19654 **CHANGE HISTORY**

19655 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19656 NAME

19657 if_nametoindex — map a network interface name to its corresponding index

19658 SYNOPSIS

19659 #include <net/if.h>

19660 unsigned if_nametoindex(const char *ifname);

19661 DESCRIPTION

19662 The *if_nametoindex()* function shall return the interface index corresponding to name *ifname*.

19663 RETURN VALUE

19664 The corresponding index if *ifname* is the name of an interface; otherwise, zero.

19665 ERRORS

19666 No errors are defined.

19667 EXAMPLES

19668 None.

19669 APPLICATION USAGE

19670 None.

19671 RATIONALE

19672 None.

19673 FUTURE DIRECTIONS

19674 None.

19675 SEE ALSO

19676 *getsockopt()*, *if_freenameindex()*, *if_indextoname()*, *if_nameindex()*, *setsockopt()*, the Base
19677 Definitions volume of IEEE Std 1003.1-2001, <net/if.h>

19678 CHANGE HISTORY

19679 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19680 **NAME**

19681 ilogb, ilogbf, ilogbl — return an unbiased exponent

19682 **SYNOPSIS**

19683 #include <math.h>

19684 int ilogb(double x);

19685 int ilogbf(float x);

19686 int ilogbl(long double x);

19687 **DESCRIPTION**

19688 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19689 conflict between the requirements described here and the ISO C standard is unintentional. This
 19690 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19691 These functions shall return the exponent part of their argument *x*. Formally, the return value is
 19692 the integral part of $\log_r |x|$ as a signed integral value, for non-zero *x*, where *r* is the radix of the
 19693 machine's floating-point arithmetic, which is the value of FLT_RADIX defined in <float.h>.

19694 An application wishing to check for error situations should set *errno* to zero and call
 19695 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 19696 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 19697 zero, an error has occurred.

19698 **RETURN VALUE**

19699 Upon successful completion, these functions shall return the exponent part of *x* as a signed
 19700 integer value. They are equivalent to calling the corresponding *logb*() function and casting the
 19701 returned value to type **int**.

19702 XSI If *x* is 0, a domain error shall occur, and the value FP_ILOGB0 shall be returned.

19703 XSI If *x* is $\pm\text{Inf}$, a domain error shall occur, and the value {INT_MAX} shall be returned.

19704 XSI If *x* is a NaN, a domain error shall occur, and the value FP_ILOGBNAN shall be returned.

19705 XSI If the correct value is greater than {INT_MAX}, {INT_MAX} shall be returned and a domain error
 19706 shall occur.

19707 If the correct value is less than {INT_MIN}, {INT_MIN} shall be returned and a domain error
 19708 shall occur.

19709 **ERRORS**

19710 These functions shall fail if:

19711 XSI	Domain Error	The <i>x</i> argument is zero, NaN, or $\pm\text{Inf}$, or the correct value is not representable as an integer.

19713	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.
19714	
19715	
19716	

19717 **EXAMPLES**

19718 None.

19719 **APPLICATION USAGE**

19720 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
19721 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19722 **RATIONALE**

19723 The errors come from taking the expected floating-point value and converting it to **int**, which is
19724 an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not
19725 representable in a type **int**), so should be a domain error.

19726 There are no known implementations that overflow. For overflow to happen, {INT_MAX} must
19727 be less than $\text{LDBL\_MAX\_EXP} * \log_2(\text{FLT\_RADIX})$ or {INT_MIN} must be greater than
19728 $\text{LDBL\_MIN\_EXP} * \log_2(\text{FLT\_RADIX})$ if subnormals are not supported, or {INT_MIN} must be
19729 greater than $(\text{LDBL\_MIN\_EXP} - \text{LDBL\_MANT\_DIG}) * \log_2(\text{FLT\_RADIX})$ if subnormals are
19730 supported.

19731 **FUTURE DIRECTIONS**

19732 None.

19733 **SEE ALSO**

19734 *feclearexcept()*, *fetestexcept()*, *logb()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-2001,
19735 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<float.h>**, **<math.h>**

19736 **CHANGE HISTORY**

19737 First released in Issue 4, Version 2.

19738 **Issue 5**

19739 Moved from X/OPEN UNIX extension to BASE.

19740 **Issue 6**19741 The *ilogb()* function is no longer marked as an extension.

19742 The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999
19743 standard.

19744 The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

19745 XSI extensions are marked.

19746 **NAME**

19747 imaxabs — return absolute value

19748 **SYNOPSIS**

19749 #include <inttypes.h>

19750 intmax_t imaxabs(intmax_t j);

19751 **DESCRIPTION**

19752 cx The functionality described on this reference page is aligned with the ISO C standard. Any
19753 conflict between the requirements described here and the ISO C standard is unintentional. This
19754 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19755 The *imaxabs()* function shall compute the absolute value of an integer *j*. If the result cannot be
19756 represented, the behavior is undefined.

19757 **RETURN VALUE**19758 The *imaxabs()* function shall return the absolute value.19759 **ERRORS**

19760 No errors are defined.

19761 **EXAMPLES**

19762 None.

19763 **APPLICATION USAGE**

19764 The absolute value of the most negative number cannot be represented in two's complement.

19765 **RATIONALE**

19766 None.

19767 **FUTURE DIRECTIONS**

19768 None.

19769 **SEE ALSO**19770 *imaxdiv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <inttypes.h>19771 **CHANGE HISTORY**

19772 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

19773 **NAME**

19774 imaxdiv — return quotient and remainder

19775 **SYNOPSIS**

19776 #include <inttypes.h>

19777 imaxdiv_t imaxdiv(intmax_t *numer*, intmax_t *denom*);19778 **DESCRIPTION**

19779 CX The functionality described on this reference page is aligned with the ISO C standard. Any
19780 conflict between the requirements described here and the ISO C standard is unintentional. This
19781 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19782 The *imaxdiv()* function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.19783 **RETURN VALUE**

19784 The *imaxdiv()* function shall return a structure of type **imaxdiv_t**, comprising both the quotient
19785 and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)
19786 and *rem* (the remainder), each of which has type **intmax_t**.

19787 If either part of the result cannot be represented, the behavior is undefined.

19788 **ERRORS**

19789 No errors are defined.

19790 **EXAMPLES**

19791 None.

19792 **APPLICATION USAGE**

19793 None.

19794 **RATIONALE**

19795 None.

19796 **FUTURE DIRECTIONS**

19797 None.

19798 **SEE ALSO**19799 *imaxabs()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**inttypes.h**>19800 **CHANGE HISTORY**

19801 First released in Issue 6. Derived from the ISO / IEC 9899:1999 standard.

19802 **NAME**19803 index — character string operations (**LEGACY**)19804 **SYNOPSIS**

19805 xSI #include <strings.h>

19806 char *index(const char *s, int c);

19807

19808 **DESCRIPTION**19809 The *index()* function shall be equivalent to *strchr()*.19810 **RETURN VALUE**19811 See *strchr()*.19812 **ERRORS**19813 See *strchr()*.19814 **EXAMPLES**

19815 None.

19816 **APPLICATION USAGE**19817 The *strchr()* function is preferred over this function.19818 For maximum portability, it is recommended to replace the function call to *index()* as follows:

19819 #define index(a,b) strchr((a),(b))

19820 **RATIONALE**

19821 None.

19822 **FUTURE DIRECTIONS**

19823 This function may be withdrawn in a future version.

19824 **SEE ALSO**19825 *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**strings.h**>19826 **CHANGE HISTORY**

19827 First released in Issue 4, Version 2.

19828 **Issue 5**

19829 Moved from X/OPEN UNIX extension to BASE.

19830 **Issue 6**

19831 This function is marked LEGACY.

19832 **NAME**

19833 inet_addr, inet_ntoa — IPv4 address manipulation

19834 **SYNOPSIS**

19835 #include <arpa/inet.h>

19836 in_addr_t inet_addr(const char *cp);

19837 char *inet_ntoa(struct in_addr in);

19838 **DESCRIPTION**19839 The *inet_addr()* function shall convert the string pointed to by *cp*, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.19841 The *inet_ntoa()* function shall convert the Internet host address specified by *in* to a string in the Internet standard dot notation.19843 The *inet_ntoa()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

19845 All Internet addresses shall be returned in network order (bytes ordered from left to right).

19846 Values specified using IPv4 dotted decimal notation take one of the following forms:

19847 a.b.c.d When four parts are specified, each shall be interpreted as a byte of data and
19848 assigned, from left to right, to the four bytes of an Internet address.19849 a.b.c When a three-part address is specified, the last part shall be interpreted as a 16-bit
19850 quantity and placed in the rightmost two bytes of the network address. This makes
19851 the three-part address format convenient for specifying Class B network addresses
19852 as "128.net.host".19853 a.b When a two-part address is supplied, the last part shall be interpreted as a 24-bit
19854 quantity and placed in the rightmost three bytes of the network address. This
19855 makes the two-part address format convenient for specifying Class A network
19856 addresses as "net.host".19857 a When only one part is given, the value shall be stored directly in the network
19858 address without any byte rearrangement.19859 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or
19860 hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal;
19861 otherwise, a leading '0' implies octal; otherwise, the number is interpreted as decimal).19862 **RETURN VALUE**19863 Upon successful completion, *inet_addr()* shall return the Internet address. Otherwise, it shall
19864 return (**in_addr_t**)(-1).19865 The *inet_ntoa()* function shall return a pointer to the network address in Internet standard dot
19866 notation.19867 **ERRORS**

19868 No errors are defined.

19869 **EXAMPLES**

19870 None.

19871 **APPLICATION USAGE**

19872 The return value of *inet_ntoa()* may point to static data that may be overwritten by subsequent
19873 calls to *inet_ntoa()*.

19874 **RATIONALE**

19875 None.

19876 **FUTURE DIRECTIONS**

19877 None.

19878 **SEE ALSO**19879 *endhostent()*, *endnetent()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**arpa/inet.h**>19880 **CHANGE HISTORY**

19881 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19882 **NAME**

19883 inet_ntop, inet_pton — convert IPv4 and IPv6 addresses between binary and text form

19884 **SYNOPSIS**

19885 #include <arpa/inet.h>

```
19886       const char *inet_ntop(int af, const void *restrict src,
19887                              char *restrict dst, socklen_t size);
19888       int inet_pton(int af, const char *restrict src, void *restrict dst);
```

19889 **DESCRIPTION**

19890 The *inet_ntop()* function shall convert a numeric address into a text string suitable for
 19891 IP6 presentation. The *af* argument shall specify the family of the address. This can be AF_INET or
 19892 AF_INET6. The *src* argument points to a buffer holding an IPv4 address if the *af* argument is
 19893 IP6 AF_INET, or an IPv6 address if the *af* argument is AF_INET6; the address must be in network
 19894 byte order. The *dst* argument points to a buffer where the function stores the resulting text
 19895 string; it shall not be NULL. The *size* argument specifies the size of this buffer, which shall be
 19896 IP6 large enough to hold the text string (INET_ADDRSTRLEN characters for IPv4,
 19897 INET6_ADDRSTRLEN characters for IPv6).

19898 The *inet_pton()* function shall convert an address in its standard text presentation form into its
 19899 IP6 numeric binary form. The *af* argument shall specify the family of the address. The AF_INET and
 19900 AF_INET6 address families shall be supported. The *src* argument points to the string being
 19901 passed in. The *dst* argument points to a buffer into which the function stores the numeric
 19902 IP6 address; this shall be large enough to hold the numeric address (32 bits for AF_INET, 128 bits for
 19903 AF_INET6).

19904 If the *af* argument of *inet_pton()* is AF_INET, the *src* string shall be in the standard IPv4 dotted-
 19905 decimal form:

19906 ddd.ddd.ddd.ddd

19907 where "ddd" is a one to three digit decimal number between 0 and 255 (see *inet_addr()*). The
 19908 *inet_pton()* function does not accept other formats (such as the octal numbers, hexadecimal
 19909 numbers, and fewer than four numbers that *inet_addr()* accepts).

19910 IP6 If the *af* argument of *inet_pton()* is AF_INET6, the *src* string shall be in one of the following
 19911 standard IPv6 text forms:

19912 1. The preferred form is "x:x:x:x:x:x:x:x", where the 'x's are the hexadecimal values
 19913 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be omitted,
 19914 but there shall be at least one numeral in every field.

19915 2. A string of contiguous zero fields in the preferred form can be shown as "::". The "::" can only appear once in an address. Unspecified addresses ("0:0:0:0:0:0:0:0") may be represented simply as "::".

19918 3. A third form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is "x:x:x:x:x:x.d.d.d.d", where the 'x's are the hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation).

19923 **Note:** A more extensive description of the standard representations of IPv6 addresses can be found in
 19924 RFC 2373.

19925

19926 **RETURN VALUE**

19927 The *inet_ntop()* function shall return a pointer to the buffer containing the text string if the
 19928 conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

19929 The *inet_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by
 19930 *IP6* *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or
 19931 a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is
 19932 unknown.

19933 **ERRORS**

19934 The *inet_ntop()* and *inet_pton()* functions shall fail if:

19935 [EAFNOSUPPORT]

19936 The *af* argument is invalid.

19937 [ENOSPC] The size of the *inet_ntop()* result buffer is inadequate.

19938 **EXAMPLES**

19939 None.

19940 **APPLICATION USAGE**

19941 None.

19942 **RATIONALE**

19943 None.

19944 **FUTURE DIRECTIONS**

19945 None.

19946 **SEE ALSO**

19947 The Base Definitions volume of IEEE Std 1003.1-2001, <**arpa/inet.h**>

19948 **CHANGE HISTORY**

19949 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19950 IPv6 extensions are marked.

19951 The **restrict** keyword is added to the *inet_ntop()* and *inet_pton()* prototypes for alignment with
 19952 the ISO/IEC 9899:1999 standard.

19953 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding “the address must
 19954 be in network byte order” to the end of the fourth sentence of the first paragraph in the
 19955 DESCRIPTION.

19956 **NAME**

19957 initstate, random, setstate, srandom — pseudo-random number functions

19958 **SYNOPSIS**

```

19959 xSI      #include <stdlib.h>

19960          char *initstate(unsigned seed, char *state, size_t size);
19961          long random(void);
19962          char *setstate(const char *state);
19963          void srandom(unsigned seed);
19964

```

19965 **DESCRIPTION**

19966 The *random()* function shall use a non-linear additive feedback random-number generator
 19967 employing a default state array size of 31 **long** integers to return successive pseudo-random
 19968 numbers in the range from 0 to $2^{31}-1$. The period of this random-number generator is
 19969 approximately $16 \times (2^{31}-1)$. The size of the state array determines the period of the random-
 19970 number generator. Increasing the state array size shall increase the period.

19971 With 256 bytes of state information, the period of the random-number generator shall be greater
 19972 than 2^{69} .

19973 Like *rand()*, *random()* shall produce by default a sequence of numbers that can be duplicated by
 19974 calling *srandom()* with 1 as the seed.

19975 The *srandom()* function shall initialize the current state array using the value of *seed*.

19976 The *initstate()* and *setstate()* functions handle restarting and changing random-number
 19977 generators. The *initstate()* function allows a state array, pointed to by the *state* argument, to be
 19978 initialized for future use. The *size* argument, which specifies the size in bytes of the state array,
 19979 shall be used by *initstate()* to decide what type of random-number generator to use; the larger
 19980 the state array, the more random the numbers. Values for the amount of state information are 8,
 19981 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one
 19982 of these values. If *initstate()* is called with $8 \leq \text{size} < 32$, then *random()* shall use a simple linear
 19983 congruential random number generator. The *seed* argument specifies a starting point for the
 19984 random-number sequence and provides for restarting at the same point. The *initstate()* function
 19985 shall return a pointer to the previous state information array.

19986 If *initstate()* has not been called, then *random()* shall behave as though *initstate()* had been called
 19987 with *seed*=1 and *size*=128.

19988 Once a state has been initialized, *setstate()* allows switching between state arrays. The array
 19989 defined by the *state* argument shall be used for further random-number generation until
 19990 *initstate()* is called or *setstate()* is called again. The *setstate()* function shall return a pointer to the
 19991 previous state array.

19992 **RETURN VALUE**

19993 If *initstate()* is called with *size* less than 8, it shall return NULL.

19994 The *random()* function shall return the generated pseudo-random number.

19995 The *srandom()* function shall not return a value.

19996 Upon successful completion, *initstate()* and *setstate()* shall return a pointer to the previous state
 19997 array; otherwise, a null pointer shall be returned.

19998 **ERRORS**

19999 No errors are defined.

20000 **EXAMPLES**

20001 None.

20002 **APPLICATION USAGE**

20003 After initialization, a state array can be restarted at a different point in one of two ways:

- 20004 1. The *initstate()* function can be used, with the desired seed, state array, and size of the
20005 array.
- 20006 2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with the
20007 desired seed. The advantage of using both of these functions is that the size of the state
20008 array does not have to be saved once it is initialized.

20009 Although some implementations of *random()* have written messages to standard error, such
20010 implementations do not conform to IEEE Std 1003.1-2001.

20011 Issue 5 restored the historical behavior of this function.

20012 Threaded applications should use *erand48()*, *nrand48()*, or *jrand48()* instead of *random()* when
20013 an independent random number sequence in multiple threads is required.

20014 **RATIONALE**

20015 None.

20016 **FUTURE DIRECTIONS**

20017 None.

20018 **SEE ALSO**20019 *drand48()*, *rand()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>20020 **CHANGE HISTORY**

20021 First released in Issue 4, Version 2.

20022 **Issue 5**

20023 Moved from X/OPEN UNIX extension to BASE.

20024 In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than
20025 or equal to 8, or less than 32”, “*size*<8” is replaced with “ $8 \leq \textit{size} < 32$ ”, and a new first paragraph
20026 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE
20027 indicating that these changes restore the historical behavior of the function.

20028 **Issue 6**

20029 In the DESCRIPTION, duplicate text “For values greater than or equal to 8...” is removed.

20030 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand_r()* from the
20031 list of suggested functions in the APPLICATION USAGE section.

20032 **NAME**

20033 insque, remque — insert or remove an element in a queue

20034 **SYNOPSIS**

```
20035 xsi      #include <search.h>

20036         void insque(void *element, void *pred);
20037         void remque(void *element);
20038
```

20039 **DESCRIPTION**

20040 The *insque()* and *remque()* functions shall manipulate queues built from doubly-linked lists. The
 20041 queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it
 20042 defines a structure in which the first two members of the structure are pointers to the same type
 20043 of structure, and any further members are application-specific. The first member of the structure
 20044 is a forward pointer to the next entry in the queue. The second member is a backward pointer to
 20045 the previous entry in the queue. If the queue is linear, the queue is terminated with null
 20046 pointers. The names of the structure and of the pointer members are not subject to any special
 20047 restriction.

20048 The *insque()* function shall insert the element pointed to by *element* into a queue immediately
 20049 after the element pointed to by *pred*.

20050 The *remque()* function shall remove the element pointed to by *element* from a queue.

20051 If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the
 20052 initial element of the queue, shall initialize the forward and backward pointers of *element* to null
 20053 pointers.

20054 If the queue is to be used as a circular list, the application shall ensure it initializes the forward
 20055 pointer and the backward pointer of the initial element of the queue to the element's own
 20056 address.

20057 **RETURN VALUE**

20058 The *insque()* and *remque()* functions do not return a value.

20059 **ERRORS**

20060 No errors are defined.

20061 **EXAMPLES**20062 **Creating a Linear Linked List**

20063 The following example creates a linear linked list.

```
20064         #include <search.h>
20065         ...
20066         struct myque element1;
20067         struct myque element2;

20068         char *data1 = "DATA1";
20069         char *data2 = "DATA2";
20070         ...
20071         element1.data = data1;
20072         element2.data = data2;

20073         insque (&element1, NULL);
20074         insque (&element2, &element1);
```


Creating a Circular Linked List

The following example creates a circular linked list.

```
#include <search.h>
...
struct myque element1;
struct myque element2;

char *data1 = "DATA1";
char *data2 = "DATA2";
...
element1.data = data1;
element2.data = data2;

element1.fwd = &element1;
element1.bck = &element1;

insque (&element2, &element1);
```

Removing an Element

The following example removes the element pointed to by *element1*.

```
#include <search.h>
...
struct myque element1;
...
remque (&element1);
```

APPLICATION USAGE

The historical implementations of these functions described the arguments as being of type **struct qelem *** rather than as being of type **void *** as defined here. In those implementations, **struct qelem** was commonly defined in **<search.h>** as:

```
struct qelem {
    struct qelem  *q_forw;
    struct qelem  *q_back;
};
```

Applications using these functions, however, were never able to use this structure directly since it provided no room for the actual data contained in the elements. Most applications defined structures that contained the two pointers as the initial elements and also provided space for, or pointers to, the object's data. Applications that used these functions to update more than one type of table also had the problem of specifying two or more different structures with the same name, if they literally used **struct qelem** as specified.

As described here, the implementations were actually expecting a structure type where the first two members were forward and backward pointers to structures. With C compilers that didn't provide function prototypes, applications used structures as specified in the DESCRIPTION above and the compiler did what the application expected.

If this method had been carried forward with an ISO C standard compiler and the historical function prototype, most applications would have to be modified to cast pointers to the structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By specifying **void *** as the argument type, applications do not need to change (unless they specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

20119 **RATIONALE**

20120 None.

20121 **FUTURE DIRECTIONS**

20122 None.

20123 **SEE ALSO**20124 The Base Definitions volume of IEEE Std 1003.1-2001, <**search.h**>20125 **CHANGE HISTORY**

20126 First released in Issue 4, Version 2.

20127 **Issue 5**

20128 Moved from X/OPEN UNIX extension to BASE.

20129 **Issue 6**

20130 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20131 **NAME**20132 **ioctl** — control a STREAMS device (**STREAMS**)20133 **SYNOPSIS**20134 **xsr** #include <stropts.h>20135 int ioctl(int *fildev*, int *request*, ... /* *arg* */);

20136

20137 **DESCRIPTION**

20138 The *ioctl()* function shall perform a variety of control functions on STREAMS devices. For non-
 20139 STREAMS devices, the functions performed by this call are unspecified. The *request* argument
 20140 and an optional third argument (with varying type) shall be passed to and interpreted by the
 20141 appropriate part of the STREAM associated with *fildev*.

20142 The *fildev* argument is an open file descriptor that refers to a device.

20143 The *request* argument selects the control function to be performed and shall depend on the
 20144 STREAMS device being addressed.

20145 The *arg* argument represents additional information that is needed by this specific STREAMS
 20146 device to perform the requested function. The type of *arg* depends upon the particular control
 20147 request, but it shall be either an integer or a pointer to a device-specific data structure.

20148 The *ioctl()* commands applicable to STREAMS, their arguments, and error conditions that apply
 20149 to each individual command are described below.

20150 The following *ioctl()* commands, with error values indicated, are applicable to all STREAMS
 20151 files:

20152 **I_PUSH** Pushes the module whose name is pointed to by *arg* onto the top of the
 20153 current STREAM, just below the STREAM head. It then calls the *open()*
 20154 function of the newly-pushed module.

20155 The *ioctl()* function with the **I_PUSH** command shall fail if:

20156 [EINVAL] Invalid module name.

20157 [ENXIO] Open function of new module failed.

20158 [ENXIO] Hangup received on *fildev*.

20159 **I_POP** Removes the module just below the STREAM head of the STREAM pointed to
 20160 by *fildev*. The *arg* argument should be 0 in an **I_POP** request.

20161 The *ioctl()* function with the **I_POP** command shall fail if:

20162 [EINVAL] No module present in the STREAM.

20163 [ENXIO] Hangup received on *fildev*.

20164 **I_LOOK** Retrieves the name of the module just below the STREAM head of the
 20165 STREAM pointed to by *fildev*, and places it in a character string pointed to by
 20166 *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long,
 20167 where FMNAMESZ is defined in <stropts.h>.

20168 The *ioctl()* function with the **I_LOOK** command shall fail if:

20169 [EINVAL] No module present in the STREAM.

20170 **I_FLUSH** Flushes read and/or write queues, depending on the value of *arg*. Valid *arg*
 20171 values are:

20172		FLUSHR	Flush all read queues.
20173		FLUSHW	Flush all write queues.
20174		FLUSHRW	Flush all read and all write queues.
20175		The <i>ioctl()</i> function with the <i>I_FLUSH</i> command shall fail if:	
20176		[EINVAL]	Invalid <i>arg</i> value.
20177		[EAGAIN] or [ENOSR]	Unable to allocate buffers for flush message.
20178			
20179		[ENXIO]	Hangup received on <i>fildev</i> .
20180	I_FLUSHBAND	Flushes a particular band of messages. The <i>arg</i> argument points to a bandinfo structure. The <i>bi_flag</i> member may be one of FLUSHR, FLUSHW, or FLUSHRW as described above. The <i>bi_pri</i> member determines the priority band to be flushed.	
20181			
20182			
20183			
20184	I_SETSIG	Requests that the STREAMS implementation send the SIGPOLL signal to the calling process when a particular event has occurred on the STREAM associated with <i>fildev</i> . I_SETSIG supports an asynchronous processing capability in STREAMS. The value of <i>arg</i> is a bitmask that specifies the events for which the process should be signaled. It is the bitwise-inclusive OR of any combination of the following constants:	
20185			
20186			
20187			
20188			
20189			
20190		S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
20191			
20192			
20193		S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
20194			
20195			
20196		S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
20197			
20198			
20199		S_HIPRI	A high-priority message is present on a STREAM head read queue. A signal shall be generated even if the message is of zero length.
20200			
20201			
20202		S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
20203			
20204			
20205			
20206		S_WRNORM	Equivalent to S_OUTPUT.
20207		S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) priority data downstream.
20208			
20209			
20210			
20211		S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the STREAM head read queue.
20212			
20213			
20214		S_ERROR	Notification of an error condition has reached the STREAM head.
20215			

20216	S_HANGUP	Notification of a hangup has reached the STREAM head.
20217	S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
20218		
20219		
20220		If <i>arg</i> is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with <i>fildev</i> .
20221		
20222		
20223		Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.
20224		
20225		
20226		The <i>ioctl()</i> function with the I_SETSIG command shall fail if:
20227	[EINVAL]	The value of <i>arg</i> is invalid.
20228	[EINVAL]	The value of <i>arg</i> is 0 and the calling process is not registered to receive the SIGPOLL signal.
20229		
20230	[EAGAIN]	There were insufficient resources to store the signal request.
20231	I_GETSIG	Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an int pointed to by <i>arg</i> , where the events are those specified in the description of I_SETSIG above.
20232		
20233		
20234		
20235		The <i>ioctl()</i> function with the I_GETSIG command shall fail if:
20236	[EINVAL]	Process is not registered to receive the SIGPOLL signal.
20237	I_FIND	Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> , and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.
20238		
20239		
20240		The <i>ioctl()</i> function with the I_FIND command shall fail if:
20241	[EINVAL]	<i>arg</i> does not contain a valid module name.
20242	I_PEEK	Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to <i>getmsg()</i> except that this command does not remove the message from the queue. The <i>arg</i> argument points to a strpeek structure.
20243		
20244		
20245		
20246		The application shall ensure that the <i>maxlen</i> member in the ctlbuf and databuf structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The <i>flags</i> member may be marked RS_HIPRI or 0, as described by <i>getmsg()</i> . If the process sets <i>flags</i> to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.
20247		
20248		
20249		
20250		
20251		
20252		I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in <i>flags</i> and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, ctlbuf specifies information in the control buffer, databuf specifies information in the data buffer, and <i>flags</i> contains the value RS_HIPRI or 0.
20253		
20254		
20255		
20256		
20257		
20258	I_SRDOPT	Sets the read mode using the value of the argument <i>arg</i> . Read modes are described in <i>read()</i> . Valid <i>arg</i> flags are:
20259		

20260		RNORM	Byte-stream mode, the default.
20261		RMSGD	Message-discard mode.
20262		RMSGN	Message-nondiscard mode.
20263		The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The	
20264		bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in	
20265		the other flag overriding RNORM which is the default.	
20266		In addition, treatment of control messages by the STREAM head may be	
20267		changed by setting any of the following flags in <i>arg</i> :	
20268		RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a
20269			control part is at the front of the STREAM head read queue.
20270		RPROTDAT	Deliver the control part of a message as data when a
20271			process issues a <i>read()</i> .
20272		RPROTDIS	Discard the control part of a message, delivering any data
20273			portion, when a process issues a <i>read()</i> .
20274		The <i>ioctl()</i> function with the I_SRDOPT command shall fail if:	
20275		[EINVAL]	The <i>arg</i> argument is not valid.
20276	I_GRDOPT	Returns the current read mode setting, as described above, in an int pointed to	
20277		by the argument <i>arg</i> . Read modes are described in <i>read()</i> .	
20278	I_NREAD	Counts the number of data bytes in the data part of the first message on the	
20279		STREAM head read queue and places this value in the int pointed to by <i>arg</i> .	
20280		The return value for the command shall be the number of messages on the	
20281		STREAM head read queue. For example, if 0 is returned in <i>arg</i> , but the <i>ioctl()</i>	
20282		return value is greater than 0, this indicates that a zero-length message is next	
20283		on the queue.	
20284	I_FDINSERT	Creates a message from specified buffer(s), adds information about another	
20285		STREAM, and sends the message downstream. The message contains a	
20286		control part and an optional data part. The data and control parts to be sent	
20287		are distinguished by placement in separate buffers, as described below. The	
20288		<i>arg</i> argument points to a strfdinsert structure.	
20289		The application shall ensure that the <i>len</i> member in the ctlbuf strbuf structure	
20290		is set to the size of a t_uscalar_t plus the number of bytes of control	
20291		information to be sent with the message. The <i>fildev</i> member specifies the file	
20292		descriptor of the other STREAM, and the <i>offset</i> member, which must be	
20293		suitably aligned for use as a t_uscalar_t , specifies the offset from the start of	
20294		the control buffer where I_FDINSERT shall store a t_uscalar_t whose	
20295		interpretation is specific to the STREAM end. The application shall ensure that	
20296		the <i>len</i> member in the databuf strbuf structure is set to the number of bytes of	
20297		data information to be sent with the message, or to 0 if no data part is to be	
20298		sent.	
20299		The <i>flags</i> member specifies the type of message to be created. A normal	
20300		message is created if <i>flags</i> is set to 0, and a high-priority message is created if	
20301		<i>flags</i> is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if	
20302		the STREAM write queue is full due to internal flow control conditions. For	
20303		priority messages, I_FDINSERT does not block on this condition. For non-	
20304		priority messages, I_FDINSERT does not block when the write queue is full	

20305		and O_NONBLOCK is set. Instead, it fails and sets <i>errno</i> to [EAGAIN].
20306		I_FDINSERT also blocks, unless prevented by lack of internal resources,
20307		waiting for the availability of message blocks in the STREAM, regardless of
20308		priority or whether O_NONBLOCK has been specified. No partial message is
20309		sent.
20310		The <i>ioctl()</i> function with the I_FDINSERT command shall fail if:
20311	[EAGAIN]	A non-priority message is specified, the O_NONBLOCK
20312		flag is set, and the STREAM write queue is full due to
20313		internal flow control conditions.
20314	[EAGAIN] or [ENOSR]	
20315		Buffers cannot be allocated for the message that is to be
20316		created.
20317	[EINVAL]	One of the following:
20318		— The <i>fildev</i> member of the strfdinsert structure is not a
20319		valid, open STREAM file descriptor.
20320		— The size of a t_uscalar_t plus <i>offset</i> is greater than the <i>len</i>
20321		member for the buffer specified through ctlbuf .
20322		— The <i>offset</i> member does not specify a properly-aligned
20323		location in the data buffer.
20324		— An undefined value is stored in <i>flags</i> .
20325	[ENXIO]	Hangup received on the STREAM identified by either the
20326		<i>fildev</i> argument or the <i>fildev</i> member of the strfdinsert
20327		structure.
20328	[ERANGE]	The <i>len</i> member for the buffer specified through databuf
20329		does not fall within the range specified by the maximum
20330		and minimum packet sizes of the topmost STREAM
20331		module; or the <i>len</i> member for the buffer specified through
20332		databuf is larger than the maximum configured size of the
20333		data part of a message; or the <i>len</i> member for the buffer
20334		specified through ctlbuf is larger than the maximum
20335		configured size of the control part of a message.
20336	I_STR	Constructs an internal STREAMS <i>ioctl()</i> message from the data pointed to by
20337		<i>arg</i> , and sends that message downstream.
20338		This mechanism is provided to send <i>ioctl()</i> requests to downstream modules
20339		and drivers. It allows information to be sent with <i>ioctl()</i> , and returns to the
20340		process any information sent upstream by the downstream recipient. I_STR
20341		shall block until the system responds with either a positive or negative
20342		acknowledgement message, or until the request times out after some period of
20343		time. If the request times out, it shall fail with <i>errno</i> set to [ETIME].
20344		At most, one I_STR can be active on a STREAM. Further I_STR calls shall
20345		block until the active I_STR completes at the STREAM head. The default
20346		timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has
20347		no effect on this call.
20348		To send requests downstream, the application shall ensure that <i>arg</i> points to a
20349		strioc structure.

20350		The <i>ic_cmd</i> member is the internal <i>ioctl()</i> command intended for a downstream module or driver and <i>ic_timeout</i> is the number of seconds (−1=infinite, 0=use implementation-defined timeout interval, >0=as specified) an I_STR request shall wait for acknowledgement before timing out. <i>ic_len</i> is the number of bytes in the data argument, and <i>ic_dp</i> is a pointer to the data argument. The <i>ic_len</i> member has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the process (the buffer pointed to by <i>ic_dp</i> should be large enough to contain the maximum amount of data that any module or the driver in the STREAM can return).
20351		
20352		
20353		
20354		
20355		
20356		
20357		
20358		
20359		
20360		The STREAM head shall convert the information pointed to by the strioc
20361		structure to an internal <i>ioctl()</i> command message and send it downstream.
20362		The <i>ioctl()</i> function with the I_STR command shall fail if:
20363		[EAGAIN] or [ENOSR]
20364		Unable to allocate buffers for the <i>ioctl()</i> message.
20365		[EINVAL] The <i>ic_len</i> member is less than 0 or larger than the
20366		maximum configured size of the data part of a message, or
20367		<i>ic_timeout</i> is less than −1.
20368		[ENXIO] Hangup received on <i>fil</i> des.
20369		[ETIME] A downstream <i>ioctl()</i> timed out before acknowledgement
20370		was received.
20371		An I_STR can also fail while waiting for an acknowledgement if a message
20372		indicating an error or a hangup is received at the STREAM head. In addition,
20373		an error code can be returned in the positive or negative acknowledgement
20374		message, in the event the <i>ioctl()</i> command sent downstream fails. For these
20375		cases, I_STR shall fail with <i>errno</i> set to the value in the message.
20376	I_SWROPT	Sets the write mode using the value of the argument <i>arg</i> . Valid bit settings for
20377		<i>arg</i> are:
20378		SNDZERO Send a zero-length message downstream when a <i>write()</i> of
20379		0 bytes occurs. To not send a zero-length message when a
20380		<i>write()</i> of 0 bytes occurs, the application shall ensure that
20381		this bit is not set in <i>arg</i> (for example, <i>arg</i> would be set to 0).
20382		The <i>ioctl()</i> function with the I_SWROPT command shall fail if:
20383		[EINVAL] <i>arg</i> is not the above value.
20384	I_GWROPT	Returns the current write mode setting, as described above, in the int that is
20385		pointed to by the argument <i>arg</i> .
20386	I_SENDFD	Creates a new reference to the open file description associated with the file
20387		descriptor <i>arg</i> , and writes a message on the STREAMS-based pipe <i>fil</i> des
20388		containing this reference, together with the user ID and group ID of the calling
20389		process.
20390		The <i>ioctl()</i> function with the I_SENDFD command shall fail if:
20391		[EAGAIN] The sending STREAM is unable to allocate a message block
20392		to contain the file pointer; or the read queue of the receiving
20393		STREAM head is full and cannot accept the message sent by
20394		I_SENDFD.

20395		[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
20396		[EINVAL]	The <i>fildev</i> argument is not connected to a STREAM pipe.
20397		[ENXIO]	Hangup received on <i>fildev</i> .
20398	I_RECVFD		Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the I_SENDFD command, and allocates a new file descriptor in the calling process that refers to this open file description. The <i>arg</i> argument is a pointer to a strrecvfd data structure as defined in <stropts.h> .
20399			
20400			
20401			
20402			
20403			The <i>fd</i> member is a file descriptor. The <i>uid</i> and <i>gid</i> members are the effective user ID and effective group ID, respectively, of the sending process.
20404			
20405			If O_NONBLOCK is not set, I_RECVFD shall block until a message is present at the STREAM head. If O_NONBLOCK is set, I_RECVFD shall fail with <i>errno</i> set to [EAGAIN] if no message is present at the STREAM head.
20406			
20407			
20408			If the message at the STREAM head is a message sent by an I_SENDFD, a new file descriptor shall be allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the <i>fd</i> member of the strrecvfd structure pointed to by <i>arg</i> .
20409			
20410			
20411			
20412			The <i>ioctl()</i> function with the I_RECVFD command shall fail if:
20413		[EAGAIN]	A message is not present at the STREAM head read queue and the O_NONBLOCK flag is set.
20414			
20415		[EBADMSG]	The message at the STREAM head read queue is not a message containing a passed file descriptor.
20416			
20417		[EMFILE]	The process has the maximum number of file descriptors currently open that it is allowed.
20418			
20419		[ENXIO]	Hangup received on <i>fildev</i> .
20420	I_LIST		Allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If <i>arg</i> is a null pointer, the return value shall be the number of modules, including the driver, that are on the STREAM pointed to by <i>fildev</i> . This lets the process allocate enough space for the module names. Otherwise, it should point to a str_list structure.
20421			
20422			
20423			
20424			
20425			The <i>sl_nmods</i> member indicates the number of entries the process has allocated in the array. Upon return, the <i>sl_modlist</i> member of the str_list structure shall contain the list of module names, and the number of entries that have been filled into the <i>sl_modlist</i> array is found in the <i>sl_nmods</i> member (the number includes the number of modules including the driver). The return value from <i>ioctl()</i> shall be 0. The entries are filled in starting at the top of the STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules (<i>sl_nmods</i>) is satisfied.
20426			
20427			
20428			
20429			
20430			
20431			
20432			
20433			The <i>ioctl()</i> function with the I_LIST command shall fail if:
20434		[EINVAL]	The <i>sl_nmods</i> member is less than 1.
20435		[EAGAIN] or [ENOSR]	Unable to allocate buffers.
20436			
20437	I_ATMARK		Allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The <i>arg</i> argument determines
20438			

20439		how the checking is done when there may be multiple marked messages on
20440		the STREAM head read queue. It may take on the following values:
20441		ANYMARK Check if the message is marked.
20442		LASTMARK Check if the message is the last one marked on the queue.
20443		The bitwise-inclusive OR of the flags ANYMARK and LASTMARK is
20444		permitted.
20445		The return value shall be 1 if the mark condition is satisfied; otherwise, the
20446		value shall be 0.
20447		The <i>ioctl()</i> function with the I_ATMARK command shall fail if:
20448		[EINVAL] Invalid <i>arg</i> value.
20449	I_CKBAND	Checks if the message of a given priority band exists on the STREAM head
20450		read queue. This shall return 1 if a message of the given priority exists, 0 if no
20451		such message exists, or –1 on error. <i>arg</i> should be of type int .
20452		The <i>ioctl()</i> function with the I_CKBAND command shall fail if:
20453		[EINVAL] Invalid <i>arg</i> value.
20454	I_GETBAND	Returns the priority band of the first message on the STREAM head read
20455		queue in the integer referenced by <i>arg</i> .
20456		The <i>ioctl()</i> function with the I_GETBAND command shall fail if:
20457		[ENODATA] No message on the STREAM head read queue.
20458	I_CANPUT	Checks if a certain band is writable. <i>arg</i> is set to the priority band in question.
20459		The return value shall be 0 if the band is flow-controlled, 1 if the band is
20460		writable, or –1 on error.
20461		The <i>ioctl()</i> function with the I_CANPUT command shall fail if:
20462		[EINVAL] Invalid <i>arg</i> value.
20463	I_SETCLTIME	This request allows the process to set the time the STREAM head shall delay
20464		when a STREAM is closing and there is data on the write queues. Before
20465		closing each module or driver, if there is data on its write queue, the STREAM
20466		head shall delay for the specified amount of time to allow the data to drain. If,
20467		after the delay, data is still present, it shall be flushed. The <i>arg</i> argument is a
20468		pointer to an integer specifying the number of milliseconds to delay, rounded
20469		up to the nearest valid value. If I_SETCLTIME is not performed on a STREAM,
20470		an implementation-defined default timeout interval is used.
20471		The <i>ioctl()</i> function with the I_SETCLTIME command shall fail if:
20472		[EINVAL] Invalid <i>arg</i> value.
20473	I_GETCLTIME	Returns the close time delay in the integer pointed to by <i>arg</i> .

20474 Multiplexed STREAMS Configurations

20475 The following commands are used for connecting and disconnecting multiplexed STREAMS
20476 configurations. These commands use an implementation-defined default timeout interval.

20477 **I_LINK** Connects two STREAMs, where *fildev* is the file descriptor of the STREAM
20478 connected to the multiplexing driver, and *arg* is the file descriptor of the
20479 STREAM connected to another driver. The STREAM designated by *arg* is
20480 connected below the multiplexing driver. I_LINK requires the multiplexing
20481 driver to send an acknowledgement message to the STREAM head regarding
20482 the connection. This call shall return a multiplexer ID number (an identifier
20483 used to disconnect the multiplexer; see I_UNLINK) on success, and -1 on
20484 failure.

20485 The *ioctl()* function with the I_LINK command shall fail if:

20486 [ENXIO] Hangup received on *fildev*.
20487 [ETIME] Timeout before acknowledgement message was received at
20488 STREAM head.
20489 [EAGAIN] or [ENOSR]
20490 Unable to allocate STREAMS storage to perform the
20491 I_LINK.
20492 [EBADF] The *arg* argument is not a valid, open file descriptor.
20493 [EINVAL] The *fildev* argument does not support multiplexing; or *arg* is
20494 not a STREAM or is already connected downstream from a
20495 multiplexer; or the specified I_LINK operation would
20496 connect the STREAM head in more than one place in the
20497 multiplexed STREAM.

20498 An I_LINK can also fail while waiting for the multiplexing driver to
20499 acknowledge the request, if a message indicating an error or a hangup is
20500 received at the STREAM head of *fildev*. In addition, an error code can be
20501 returned in the positive or negative acknowledgement message. For these
20502 cases, I_LINK fails with *errno* set to the value in the message.

20503 **I_UNLINK** Disconnects the two STREAMs specified by *fildev* and *arg*. *fildev* is the file
20504 descriptor of the STREAM connected to the multiplexing driver. The *arg*
20505 argument is the multiplexer ID number that was returned by the I_LINK
20506 *ioctl()* command when a STREAM was connected downstream from the
20507 multiplexing driver. If *arg* is MUXID_ALL, then all STREAMs that were
20508 connected to *fildev* shall be disconnected. As in I_LINK, this command
20509 requires acknowledgement.

20510 The *ioctl()* function with the I_UNLINK command shall fail if:

20511 [ENXIO] Hangup received on *fildev*.
20512 [ETIME] Timeout before acknowledgement message was received at
20513 STREAM head.
20514 [EAGAIN] or [ENOSR]
20515 Unable to allocate buffers for the acknowledgement
20516 message.
20517 [EINVAL] Invalid multiplexer ID number.

20518		An I_UNLINK can also fail while waiting for the multiplexing driver to
20519		acknowledge the request if a message indicating an error or a hangup is
20520		received at the STREAM head of <i>filde</i> s. In addition, an error code can be
20521		returned in the positive or negative acknowledgement message. For these
20522		cases, I_UNLINK shall fail with <i>errno</i> set to the value in the message.
20523	I_PLINK	Creates a <i>persistent connection</i> between two STREAMs, where <i>filde</i> s is the file
20524		descriptor of the STREAM connected to the multiplexing driver, and <i>arg</i> is the
20525		file descriptor of the STREAM connected to another driver. This call shall
20526		create a persistent connection which can exist even if the file descriptor <i>filde</i> s
20527		associated with the upper STREAM to the multiplexing driver is closed. The
20528		STREAM designated by <i>arg</i> gets connected via a persistent connection below
20529		the multiplexing driver. I_PLINK requires the multiplexing driver to send an
20530		acknowledgement message to the STREAM head. This call shall return a
20531		multiplexer ID number (an identifier that may be used to disconnect the
20532		multiplexer; see I_PUNLINK) on success, and -1 on failure.
20533		The <i>ioctl()</i> function with the I_PLINK command shall fail if:
20534	[ENXIO]	Hangup received on <i>filde</i> s.
20535	[ETIME]	Timeout before acknowledgement message was received at
20536		STREAM head.
20537	[EAGAIN] or [ENOSR]	
20538		Unable to allocate STREAMS storage to perform the
20539		I_PLINK.
20540	[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
20541	[EINVAL]	The <i>filde</i> s argument does not support multiplexing; or <i>arg</i> is
20542		not a STREAM or is already connected downstream from a
20543		multiplexer; or the specified I_PLINK operation would
20544		connect the STREAM head in more than one place in the
20545		multiplexed STREAM.
20546		An I_PLINK can also fail while waiting for the multiplexing driver to
20547		acknowledge the request, if a message indicating an error or a hangup is
20548		received at the STREAM head of <i>filde</i> s. In addition, an error code can be
20549		returned in the positive or negative acknowledgement message. For these
20550		cases, I_PLINK shall fail with <i>errno</i> set to the value in the message.
20551	I_PUNLINK	Disconnects the two STREAMs specified by <i>filde</i> s and <i>arg</i> from a persistent
20552		connection. The <i>filde</i> s argument is the file descriptor of the STREAM
20553		connected to the multiplexing driver. The <i>arg</i> argument is the multiplexer ID
20554		number that was returned by the I_PLINK <i>ioctl()</i> command when a STREAM
20555		was connected downstream from the multiplexing driver. If <i>arg</i> is
20556		MUXID_ALL, then all STREAMs which are persistent connections to <i>filde</i> s
20557		shall be disconnected. As in I_PLINK, this command requires the multiplexing
20558		driver to acknowledge the request.
20559		The <i>ioctl()</i> function with the I_PUNLINK command shall fail if:
20560	[ENXIO]	Hangup received on <i>filde</i> s.
20561	[ETIME]	Timeout before acknowledgement message was received at
20562		STREAM head.

20563 [EAGAIN] or [ENOSR]
 20564 Unable to allocate buffers for the acknowledgement
 20565 message.

20566 [EINVAL] Invalid multiplexer ID number.

20567 An I_PUNLINK can also fail while waiting for the multiplexing driver to
 20568 acknowledge the request if a message indicating an error or a hangup is
 20569 received at the STREAM head of *fildev*. In addition, an error code can be
 20570 returned in the positive or negative acknowledgement message. For these
 20571 cases, I_PUNLINK shall fail with *errno* set to the value in the message.

20572 RETURN VALUE

20573 Upon successful completion, *ioctl()* shall return a value other than -1 that depends upon the
 20574 STREAMS device control function. Otherwise, it shall return -1 and set *errno* to indicate the
 20575 error.

20576 ERRORS

20577 Under the following general conditions, *ioctl()* shall fail if:

20578 [EBADF] The *fildev* argument is not a valid open file descriptor.

20579 [EINTR] A signal was caught during the *ioctl()* operation.

20580 [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or
 20581 indirectly) downstream from a multiplexer.

20582 If an underlying device driver detects an error, then *ioctl()* shall fail if:

20583 [EINVAL] The *request* or *arg* argument is not valid for this device.

20584 [EIO] Some physical I/O error has occurred.

20585 [ENOTTY] The *fildev* argument is not associated with a STREAMS device that accepts
 20586 control functions.

20587 [ENXIO] The *request* and *arg* arguments are valid for this device driver, but the service
 20588 requested cannot be performed on this particular sub-device.

20589 [ENODEV] The *fildev* argument refers to a valid STREAMS device, but the corresponding
 20590 device driver does not support the *ioctl()* function.

20591 If a STREAM is connected downstream from a multiplexer, any *ioctl()* command except
 20592 I_UNLINK and I_PUNLINK shall set *errno* to [EINVAL].

20593 EXAMPLES

20594 None.

20595 APPLICATION USAGE

20596 The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

20597 RATIONALE

20598 None.

20599 FUTURE DIRECTIONS

20600 None.

20601 SEE ALSO

20602 Section 2.6 (on page 38), *close()*, *fcntl()*, *getmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *read()*,
 20603 *sigaction()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stropts.h**>

20604 **CHANGE HISTORY**

20605 First released in Issue 4, Version 2.

20606 **Issue 5**

20607 Moved from X/OPEN UNIX extension to BASE.

20608 **Issue 6**

20609 The Open Group Corrigendum U028/4 is applied, correcting text in the `I_FDINSERT [EINVAL]`
 20610 case to refer to *ctlbuf*.

20611 This function is marked as part of the XSI STREAMS Option Group.

20612 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20613 **NAME**

20614 isalnum — test for an alphanumeric character

20615 **SYNOPSIS**

20616 #include <ctype.h>

20617 int isalnum(int c);

20618 **DESCRIPTION**

20619 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20620 conflict between the requirements described here and the ISO C standard is unintentional. This
20621 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20622 The *isalnum()* function shall test whether *c* is a character of class **alpha** or **digit** in the program's
20623 current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20624 The *c* argument is an **int**, the value of which the application shall ensure is representable as an
20625 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the
20626 behavior is undefined.

20627 **RETURN VALUE**

20628 The *isalnum()* function shall return non-zero if *c* is an alphanumeric character; otherwise, it shall
20629 return 0.

20630 **ERRORS**

20631 No errors are defined.

20632 **EXAMPLES**

20633 None.

20634 **APPLICATION USAGE**

20635 To ensure applications portability, especially across natural languages, only this function and
20636 those listed in the SEE ALSO section should be used for character classification.

20637 **RATIONALE**

20638 None.

20639 **FUTURE DIRECTIONS**

20640 None.

20641 **SEE ALSO**

20642 *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*,
20643 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>,
20644 <stdio.h>

20645 **CHANGE HISTORY**

20646 First released in Issue 1. Derived from Issue 1 of the SVID.

20647 **Issue 6**

20648 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20649 **NAME**

20650 isalpha — test for an alphabetic character

20651 **SYNOPSIS**

20652 #include <ctype.h>

20653 int isalpha(int c);

20654 **DESCRIPTION**

20655 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20656 conflict between the requirements described here and the ISO C standard is unintentional. This
20657 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20658 The *isalpha()* function shall test whether *c* is a character of class **alpha** in the program's current
20659 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20660 The *c* argument is an **int**, the value of which the application shall ensure is representable as an
20661 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the
20662 behavior is undefined.

20663 **RETURN VALUE**

20664 The *isalpha()* function shall return non-zero if *c* is an alphabetic character; otherwise, it shall
20665 return 0.

20666 **ERRORS**

20667 No errors are defined.

20668 **EXAMPLES**

20669 None.

20670 **APPLICATION USAGE**

20671 To ensure applications portability, especially across natural languages, only this function and
20672 those listed in the SEE ALSO section should be used for character classification.

20673 **RATIONALE**

20674 None.

20675 **FUTURE DIRECTIONS**

20676 None.

20677 **SEE ALSO**

20678 *isalnum()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
20679 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
20680 <ctype.h>, <stdio.h>

20681 **CHANGE HISTORY**

20682 First released in Issue 1. Derived from Issue 1 of the SVID.

20683 **Issue 6**

20684 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20685 **NAME**

20686 isascii — test for a 7-bit US-ASCII character

20687 **SYNOPSIS**

20688 xSI #include <ctype.h>

20689 int isascii(int c);

20690

20691 **DESCRIPTION**20692 The *isascii()* function shall test whether *c* is a 7-bit US-ASCII character code.20693 The *isascii()* function is defined on all integer values.20694 **RETURN VALUE**20695 The *isascii()* function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and
20696 octal 0177 inclusive; otherwise, it shall return 0.20697 **ERRORS**

20698 No errors are defined.

20699 **EXAMPLES**

20700 None.

20701 **APPLICATION USAGE**

20702 None.

20703 **RATIONALE**

20704 None.

20705 **FUTURE DIRECTIONS**

20706 None.

20707 **SEE ALSO**

20708 The Base Definitions volume of IEEE Std 1003.1-2001, <ctype.h>

20709 **CHANGE HISTORY**

20710 First released in Issue 1. Derived from Issue 1 of the SVID.

20711 **NAME**

20712 isastream — test a file descriptor (**STREAMS**)

20713 **SYNOPSIS**

20714 xSR `#include <stropts.h>`

20715 `int isastream(int fildes);`

20716

20717 **DESCRIPTION**

20718 The *isastream()* function shall test whether *fildes*, an open file descriptor, is associated with a
20719 STREAMS-based file.

20720 **RETURN VALUE**

20721 Upon successful completion, *isastream()* shall return 1 if *fildes* refers to a STREAMS-based file
20722 and 0 if not. Otherwise, *isastream()* shall return -1 and set *errno* to indicate the error.

20723 **ERRORS**

20724 The *isastream()* function shall fail if:

20725 [EBADF] The *fildes* argument is not a valid open file descriptor.

20726 **EXAMPLES**

20727 None.

20728 **APPLICATION USAGE**

20729 None.

20730 **RATIONALE**

20731 None.

20732 **FUTURE DIRECTIONS**

20733 None.

20734 **SEE ALSO**

20735 The Base Definitions volume of IEEE Std 1003.1-2001, <**stropts.h**>

20736 **CHANGE HISTORY**

20737 First released in Issue 4, Version 2.

20738 **Issue 5**

20739 Moved from X/OPEN UNIX extension to BASE.

20740 **NAME**

20741 isatty — test for a terminal device

20742 **SYNOPSIS**

20743 #include <unistd.h>

20744 int isatty(int *fildev*);20745 **DESCRIPTION**20746 The *isatty()* function shall test whether *fildev*, an open file descriptor, is associated with a
20747 terminal device.20748 **RETURN VALUE**20749 The *isatty()* function shall return 1 if *fildev* is associated with a terminal; otherwise, it shall return
20750 0 and may set *errno* to indicate the error.20751 **ERRORS**20752 The *isatty()* function may fail if:20753 [EBADF] The *fildev* argument is not a valid open file descriptor.20754 [ENOTTY] The *fildev* argument is not associated with a terminal.20755 **EXAMPLES**

20756 None.

20757 **APPLICATION USAGE**20758 The *isatty()* function does not necessarily indicate that a human being is available for interaction
20759 via *fildev*. It is quite possible that non-terminal devices are connected to the communications
20760 line.20761 **RATIONALE**

20762 None.

20763 **FUTURE DIRECTIONS**

20764 None.

20765 **SEE ALSO**20766 The Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>20767 **CHANGE HISTORY**

20768 First released in Issue 1. Derived from Issue 1 of the SVID.

20769 **Issue 6**20770 The following new requirements on POSIX implementations derive from alignment with the
20771 Single UNIX Specification:

- 20772
- The optional setting of *errno* to indicate an error is added.
 - The [EBADF] and [ENOTTY] optional error conditions are added.
- 20773

20774 **NAME**

20775 isblank — test for a blank character

20776 **SYNOPSIS**

20777 #include <ctype.h>

20778 int isblank(int c);

20779 **DESCRIPTION**

20780 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20781 conflict between the requirements described here and the ISO C standard is unintentional. This
20782 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20783 The *isblank()* function shall test whether *c* is a character of class **blank** in the program's current
20784 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20785 The *c* argument is a type **int**, the value of which the application shall ensure is a character
20786 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
20787 any other value, the behavior is undefined.

20788 **RETURN VALUE**20789 The *isblank()* function shall return non-zero if *c* is a <blank>; otherwise, it shall return 0.20790 **ERRORS**

20791 No errors are defined.

20792 **EXAMPLES**

20793 None.

20794 **APPLICATION USAGE**

20795 To ensure applications portability, especially across natural languages, only this function and
20796 those listed in the SEE ALSO section should be used for character classification.

20797 **RATIONALE**

20798 None.

20799 **FUTURE DIRECTIONS**

20800 None.

20801 **SEE ALSO**

20802 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
20803 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
20804 <ctype.h>

20805 **CHANGE HISTORY**

20806 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20807 **NAME**

20808 isctrl — test for a control character

20809 **SYNOPSIS**

20810 #include <ctype.h>

20811 int isctrl(int c);

20812 **DESCRIPTION**

20813 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20814 conflict between the requirements described here and the ISO C standard is unintentional. This
20815 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20816 The *isctrl()* function shall test whether *c* is a character of class **cntrl** in the program's current
20817 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20818 The *c* argument is a type **int**, the value of which the application shall ensure is a character
20819 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
20820 any other value, the behavior is undefined.

20821 **RETURN VALUE**20822 The *isctrl()* function shall return non-zero if *c* is a control character; otherwise, it shall return 0.20823 **ERRORS**

20824 No errors are defined.

20825 **EXAMPLES**

20826 None.

20827 **APPLICATION USAGE**

20828 To ensure applications portability, especially across natural languages, only this function and
20829 those listed in the SEE ALSO section should be used for character classification.

20830 **RATIONALE**

20831 None.

20832 **FUTURE DIRECTIONS**

20833 None.

20834 **SEE ALSO**

20835 *isalnum()*, *isalpha()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
20836 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
20837 <ctype.h>

20838 **CHANGE HISTORY**

20839 First released in Issue 1. Derived from Issue 1 of the SVID.

20840 **Issue 6**

20841 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20842 **NAME**

20843 isdigit — test for a decimal digit

20844 **SYNOPSIS**

20845 #include <ctype.h>

20846 int isdigit(int c);

20847 **DESCRIPTION**

20848 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20849 conflict between the requirements described here and the ISO C standard is unintentional. This
20850 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20851 The *isdigit()* function shall test whether *c* is a character of class **digit** in the program's current
20852 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20853 The *c* argument is an **int**, the value of which the application shall ensure is a character
20854 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
20855 any other value, the behavior is undefined.

20856 **RETURN VALUE**20857 The *isdigit()* function shall return non-zero if *c* is a decimal digit; otherwise, it shall return 0.20858 **ERRORS**

20859 No errors are defined.

20860 **EXAMPLES**

20861 None.

20862 **APPLICATION USAGE**

20863 To ensure applications portability, especially across natural languages, only this function and
20864 those listed in the SEE ALSO section should be used for character classification.

20865 **RATIONALE**

20866 None.

20867 **FUTURE DIRECTIONS**

20868 None.

20869 **SEE ALSO**

20870 *isalnum()*, *isalpha()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
20871 *isxdigit()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

20872 **CHANGE HISTORY**

20873 First released in Issue 1. Derived from Issue 1 of the SVID.

20874 **Issue 6**

20875 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20876 **NAME**

20877 isfinite — test for finite value

20878 **SYNOPSIS**

20879 #include <math.h>

20880 int isfinite(real-floating x);

20881 **DESCRIPTION**

20882 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20883 conflict between the requirements described here and the ISO C standard is unintentional. This
20884 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20885 The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or
20886 normal, and not infinite or NaN). First, an argument represented in a format wider than its
20887 semantic type is converted to its semantic type. Then determination is based on the type of the
20888 argument.

20889 **RETURN VALUE**20890 The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.20891 **ERRORS**

20892 No errors are defined.

20893 **EXAMPLES**

20894 None.

20895 **APPLICATION USAGE**

20896 None.

20897 **RATIONALE**

20898 None.

20899 **FUTURE DIRECTIONS**

20900 None.

20901 **SEE ALSO**

20902 *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of
20903 IEEE Std 1003.1-2001 <math.h>

20904 **CHANGE HISTORY**

20905 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20906 **NAME**

20907 isgraph — test for a visible character

20908 **SYNOPSIS**

20909 #include <ctype.h>

20910 int isgraph(int c);

20911 **DESCRIPTION**

20912 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20913 conflict between the requirements described here and the ISO C standard is unintentional. This
20914 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20915 The *isgraph()* function shall test whether *c* is a character of class **graph** in the program's current
20916 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20917 The *c* argument is an **int**, the value of which the application shall ensure is a character
20918 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
20919 any other value, the behavior is undefined.

20920 **RETURN VALUE**

20921 The *isgraph()* function shall return non-zero if *c* is a character with a visible representation;
20922 otherwise, it shall return 0.

20923 **ERRORS**

20924 No errors are defined.

20925 **EXAMPLES**

20926 None.

20927 **APPLICATION USAGE**

20928 To ensure applications portability, especially across natural languages, only this function and
20929 those listed in the SEE ALSO section should be used for character classification.

20930 **RATIONALE**

20931 None.

20932 **FUTURE DIRECTIONS**

20933 None.

20934 **SEE ALSO**

20935 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*,
20936 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

20937 **CHANGE HISTORY**

20938 First released in Issue 1. Derived from Issue 1 of the SVID.

20939 **Issue 6**

20940 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20941 **NAME**

20942 isgreater — test if x greater than y

20943 **SYNOPSIS**

20944 #include <math.h>

20945 int isgreater(real-floating x, real-floating y);

20946 **DESCRIPTION**

20947 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20948 conflict between the requirements described here and the ISO C standard is unintentional. This
20949 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20950 The *isgreater()* macro shall determine whether its first argument is greater than its second
20951 argument. The value of *isgreater(x, y)* shall be equal to $(x) > (y)$; however, unlike $(x) > (y)$,
20952 *isgreater(x, y)* shall not raise the invalid floating-point exception when x and y are unordered.

20953 **RETURN VALUE**20954 Upon successful completion, the *isgreater()* macro shall return the value of $(x) > (y)$.

20955 If x or y is NaN, 0 shall be returned.

20956 **ERRORS**

20957 No errors are defined.

20958 **EXAMPLES**

20959 None.

20960 **APPLICATION USAGE**

20961 The relational and equality operators support the usual mathematical relationships between
20962 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
20963 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
20964 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
20965 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
20966 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
20967 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
20968 indicates that the argument shall be an expression of **real-floating** type.

20969 **RATIONALE**

20970 None.

20971 **FUTURE DIRECTIONS**

20972 None.

20973 **SEE ALSO**

20974 *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of
20975 IEEE Std 1003.1-2001 <math.h>

20976 **CHANGE HISTORY**

20977 First released in Issue 6. Derived from the ISO / IEC 9899:1999 standard.

20978 **NAME**20979 isgreaterqual — test if *x* is greater than or equal to *y*20980 **SYNOPSIS**

20981 #include <math.h>

20982 int isgreaterqual(real-floating *x*, real-floating *y*);20983 **DESCRIPTION**

20984 cx The functionality described on this reference page is aligned with the ISO C standard. Any
20985 conflict between the requirements described here and the ISO C standard is unintentional. This
20986 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20987 The *isgreaterqual()* macro shall determine whether its first argument is greater than or equal to
20988 its second argument. The value of *isgreaterqual*(*x*, *y*) shall be equal to (*x*) >= (*y*); however, unlike
20989 (*x*) >= (*y*), *isgreaterqual*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are
20990 unordered.

20991 **RETURN VALUE**20992 Upon successful completion, the *isgreaterqual()* macro shall return the value of (*x*) >= (*y*).20993 If *x* or *y* is NaN, 0 shall be returned.20994 **ERRORS**

20995 No errors are defined.

20996 **EXAMPLES**

20997 None.

20998 **APPLICATION USAGE**

20999 The relational and equality operators support the usual mathematical relationships between
21000 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21001 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21002 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21003 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21004 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21005 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21006 indicates that the argument shall be an expression of **real-floating** type.

21007 **RATIONALE**

21008 None.

21009 **FUTURE DIRECTIONS**

21010 None.

21011 **SEE ALSO**

21012 *isgreater()*, *isless()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of
21013 IEEE Std 1003.1-2001 <math.h>

21014 **CHANGE HISTORY**

21015 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21016 **NAME**

21017 isinf — test for infinity

21018 **SYNOPSIS**

21019 #include <math.h>

21020 int isinf(real-floating x);

21021 **DESCRIPTION**

21022 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21023 conflict between the requirements described here and the ISO C standard is unintentional. This
21024 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21025 The *isinf()* macro shall determine whether its argument value is an infinity (positive or
21026 negative). First, an argument represented in a format wider than its semantic type is converted
21027 to its semantic type. Then determination is based on the type of the argument.

21028 **RETURN VALUE**21029 The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.21030 **ERRORS**

21031 No errors are defined.

21032 **EXAMPLES**

21033 None.

21034 **APPLICATION USAGE**

21035 None.

21036 **RATIONALE**

21037 None.

21038 **FUTURE DIRECTIONS**

21039 None.

21040 **SEE ALSO**

21041 *fpclassify()*, *isfinite()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of
21042 IEEE Std 1003.1-2001 <math.h>

21043 **CHANGE HISTORY**

21044 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21045 **NAME**21046 isless — test if *x* is less than *y*21047 **SYNOPSIS**

21048 #include <math.h>

21049 int isless(real-floating *x*, real-floating *y*);21050 **DESCRIPTION**

21051 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21052 conflict between the requirements described here and the ISO C standard is unintentional. This
21053 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21054 The *isless()* macro shall determine whether its first argument is less than its second argument.
21055 The value of *isless*(*x*, *y*) shall be equal to (*x*) < (*y*); however, unlike (*x*) < (*y*), *isless*(*x*, *y*) shall not
21056 raise the invalid floating-point exception when *x* and *y* are unordered.

21057 **RETURN VALUE**21058 Upon successful completion, the *isless()* macro shall return the value of (*x*) < (*y*).21059 If *x* or *y* is NaN, 0 shall be returned.21060 **ERRORS**

21061 No errors are defined.

21062 **EXAMPLES**

21063 None.

21064 **APPLICATION USAGE**

21065 The relational and equality operators support the usual mathematical relationships between
21066 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21067 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21068 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21069 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21070 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21071 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21072 indicates that the argument shall be an expression of **real-floating** type.

21073 **RATIONALE**

21074 None.

21075 **FUTURE DIRECTIONS**

21076 None.

21077 **SEE ALSO**

21078 *isgreater()*, *isgreaterequal()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume
21079 of IEEE Std 1003.1-2001, <math.h>

21080 **CHANGE HISTORY**

21081 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21082 **NAME**21083 islessequal — test if *x* is less than or equal to *y*21084 **SYNOPSIS**

21085 #include <math.h>

21086 int islessequal(real-floating *x*, real-floating *y*);21087 **DESCRIPTION**

21088 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
21089 conflict between the requirements described here and the ISO C standard is unintentional. This
21090 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21091 The *islessequal()* macro shall determine whether its first argument is less than or equal to its
21092 second argument. The value of *islessequal*(*x*, *y*) shall be equal to (*x*) <= (*y*); however, unlike
21093 (*x*) <= (*y*), *islessequal*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are
21094 unordered.

21095 **RETURN VALUE**21096 Upon successful completion, the *islessequal()* macro shall return the value of (*x*) <= (*y*).21097 If *x* or *y* is NaN, 0 shall be returned.21098 **ERRORS**

21099 No errors are defined.

21100 **EXAMPLES**

21101 None.

21102 **APPLICATION USAGE**

21103 The relational and equality operators support the usual mathematical relationships between
21104 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21105 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21106 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21107 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21108 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21109 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21110 indicates that the argument shall be an expression of **real-floating** type.

21111 **RATIONALE**

21112 None.

21113 **FUTURE DIRECTIONS**

21114 None.

21115 **SEE ALSO**

21116 *isgreater()*, *isgreaterequal()*, *isless()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of
21117 IEEE Std 1003.1-2001 <math.h>

21118 **CHANGE HISTORY**

21119 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21120 **NAME**

21121 islessgreater — test if *x* is less than or greater than *y*

21122 **SYNOPSIS**

21123 #include <math.h>

21124 int islessgreater(real-floating *x*, real-floating *y*);

21125 **DESCRIPTION**

21126 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21127 conflict between the requirements described here and the ISO C standard is unintentional. This
21128 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21129 The *islessgreater()* macro shall determine whether its first argument is less than or greater than
21130 its second argument. The *islessgreater*(*x*, *y*) macro is similar to $(x) < (y) \mid \mid (x) > (y)$; however,
21131 *islessgreater*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered
21132 (nor shall it evaluate *x* and *y* twice).

21133 **RETURN VALUE**

21134 Upon successful completion, the *islessgreater()* macro shall return the value of
21135 $(x) < (y) \mid \mid (x) > (y)$.

21136 If *x* or *y* is NaN, 0 shall be returned.

21137 **ERRORS**

21138 No errors are defined.

21139 **EXAMPLES**

21140 None.

21141 **APPLICATION USAGE**

21142 The relational and equality operators support the usual mathematical relationships between
21143 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21144 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21145 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21146 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21147 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21148 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21149 indicates that the argument shall be an expression of **real-floating** type.

21150 **RATIONALE**

21151 None.

21152 **FUTURE DIRECTIONS**

21153 None.

21154 **SEE ALSO**

21155 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *isunordered()*, the Base Definitions volume of
21156 IEEE Std 1003.1-2001 <math.h>

21157 **CHANGE HISTORY**

21158 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21159 **NAME**

21160 islower — test for a lowercase letter

21161 **SYNOPSIS**

21162 #include <ctype.h>

21163 int islower(int c);

21164 **DESCRIPTION**

21165 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 21166 conflict between the requirements described here and the ISO C standard is unintentional. This
 21167 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21168 The *islower()* function shall test whether *c* is a character of class **lower** in the program's current
 21169 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21170 The *c* argument is an **int**, the value of which the application shall ensure is a character
 21171 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 21172 any other value, the behavior is undefined.

21173 **RETURN VALUE**21174 The *islower()* function shall return non-zero if *c* is a lowercase letter; otherwise, it shall return 0.21175 **ERRORS**

21176 No errors are defined.

21177 **EXAMPLES**21178 **Testing for a Lowercase Letter**

21179 The following example tests whether the value is a lowercase letter, based on the locale of the
 21180 user, then uses it as part of a key value.

```

21181 #include <ctype.h>
21182 #include <stdlib.h>
21183 #include <locale.h>
21184 ...
21185 char *keyst;
21186 int elementlen, len;
21187 char c;
21188 ...
21189 setlocale(LC_ALL, "");
21190 ...
21191 len = 0;
21192 while (len < elementlen) {
21193     c = (char) (rand() % 256);
21194     ...
21195     if (islower(c))
21196         keyst[len++] = c;
21197 }
21198 ...

```

21199 **APPLICATION USAGE**

21200 To ensure applications portability, especially across natural languages, only this function and
 21201 those listed in the SEE ALSO section should be used for character classification.

21202 **RATIONALE**

21203 None.

21204 **FUTURE DIRECTIONS**

21205 None.

21206 **SEE ALSO**

21207 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
21208 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
21209 <ctype.h>

21210 **CHANGE HISTORY**

21211 First released in Issue 1. Derived from Issue 1 of the SVID.

21212 **Issue 6**

21213 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21214 An example is added.

21215 **NAME**

21216 isnan — test for a NaN

21217 **SYNOPSIS**

21218 #include <math.h>

21219 int isnan(real-floating x);

21220 **DESCRIPTION**

21221 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21222 conflict between the requirements described here and the ISO C standard is unintentional. This
21223 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21224 The *isnan()* macro shall determine whether its argument value is a NaN. First, an argument
21225 represented in a format wider than its semantic type is converted to its semantic type. Then
21226 determination is based on the type of the argument.

21227 **RETURN VALUE**21228 The *isnan()* macro shall return a non-zero value if and only if its argument has a NaN value.21229 **ERRORS**

21230 No errors are defined.

21231 **EXAMPLES**

21232 None.

21233 **APPLICATION USAGE**

21234 None.

21235 **RATIONALE**

21236 None.

21237 **FUTURE DIRECTIONS**

21238 None.

21239 **SEE ALSO**

21240 *fpclassify()*, *isfinite()*, *isinf()*, *isnormal()*, *signbit()*, the Base Definitions volume of
21241 IEEE Std 1003.1-2001, <math.h>

21242 **CHANGE HISTORY**

21243 First released in Issue 3.

21244 **Issue 5**

21245 The DESCRIPTION is updated to indicate the return value when NaN is not supported. This
21246 text was previously published in the APPLICATION USAGE section.

21247 **Issue 6**

21248 Re-written for alignment with the ISO/IEC 9899:1999 standard.

21249 **NAME**

21250 isnormal — test for a normal value

21251 **SYNOPSIS**

21252 #include <math.h>

21253 int isnormal(real-floating x);

21254 **DESCRIPTION**

21255 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21256 conflict between the requirements described here and the ISO C standard is unintentional. This
21257 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21258 The *isnormal()* macro shall determine whether its argument value is normal (neither zero,
21259 subnormal, infinite, nor NaN). First, an argument represented in a format wider than its
21260 semantic type is converted to its semantic type. Then determination is based on the type of the
21261 argument.

21262 **RETURN VALUE**

21263 The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal
21264 value.

21265 **ERRORS**

21266 No errors are defined.

21267 **EXAMPLES**

21268 None.

21269 **APPLICATION USAGE**

21270 None.

21271 **RATIONALE**

21272 None.

21273 **FUTURE DIRECTIONS**

21274 None.

21275 **SEE ALSO**

21276 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *signbit()*, the Base Definitions volume of
21277 IEEE Std 1003.1-2001, <math.h>

21278 **CHANGE HISTORY**

21279 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21280 **NAME**

21281 isprint — test for a printable character

21282 **SYNOPSIS**

21283 #include <ctype.h>

21284 int isprint(int c);

21285 **DESCRIPTION**

21286 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21287 conflict between the requirements described here and the ISO C standard is unintentional. This
21288 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21289 The *isprint()* function shall test whether *c* is a character of class **print** in the program's current
21290 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21291 The *c* argument is an **int**, the value of which the application shall ensure is a character
21292 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
21293 any other value, the behavior is undefined.

21294 **RETURN VALUE**

21295 The *isprint()* function shall return non-zero if *c* is a printable character; otherwise, it shall return
21296 0.

21297 **ERRORS**

21298 No errors are defined.

21299 **EXAMPLES**

21300 None.

21301 **APPLICATION USAGE**

21302 To ensure applications portability, especially across natural languages, only this function and
21303 those listed in the SEE ALSO section should be used for character classification.

21304 **RATIONALE**

21305 None.

21306 **FUTURE DIRECTIONS**

21307 None.

21308 **SEE ALSO**

21309 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *ispunct()*, *isspace()*, *isupper()*,
21310 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
21311 <ctype.h>

21312 **CHANGE HISTORY**

21313 First released in Issue 1. Derived from Issue 1 of the SVID.

21314 **Issue 6**

21315 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21316 **NAME**

21317 ispunct — test for a punctuation character

21318 **SYNOPSIS**

21319 #include <ctype.h>

21320 int ispunct(int c);

21321 **DESCRIPTION**

21322 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21323 conflict between the requirements described here and the ISO C standard is unintentional. This
21324 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21325 The *ispunct()* function shall test whether *c* is a character of class **punct** in the program's current
21326 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21327 The *c* argument is an **int**, the value of which the application shall ensure is a character
21328 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
21329 any other value, the behavior is undefined.

21330 **RETURN VALUE**

21331 The *ispunct()* function shall return non-zero if *c* is a punctuation character; otherwise, it shall
21332 return 0.

21333 **ERRORS**

21334 No errors are defined.

21335 **EXAMPLES**

21336 None.

21337 **APPLICATION USAGE**

21338 To ensure applications portability, especially across natural languages, only this function and
21339 those listed in the SEE ALSO section should be used for character classification.

21340 **RATIONALE**

21341 None.

21342 **FUTURE DIRECTIONS**

21343 None.

21344 **SEE ALSO**

21345 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*, *isxdigit()*,
21346 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

21347 **CHANGE HISTORY**

21348 First released in Issue 1. Derived from Issue 1 of the SVID.

21349 **Issue 6**

21350 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21351 **NAME**

21352 isspace — test for a white-space character

21353 **SYNOPSIS**

21354 #include <ctype.h>

21355 int isspace(int c);

21356 **DESCRIPTION**

21357 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21358 conflict between the requirements described here and the ISO C standard is unintentional. This
21359 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21360 The *isspace()* function shall test whether *c* is a character of class **space** in the program's current
21361 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21362 The *c* argument is an **int**, the value of which the application shall ensure is a character
21363 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
21364 any other value, the behavior is undefined.

21365 **RETURN VALUE**

21366 The *isspace()* function shall return non-zero if *c* is a white-space character; otherwise, it shall
21367 return 0.

21368 **ERRORS**

21369 No errors are defined.

21370 **EXAMPLES**

21371 None.

21372 **APPLICATION USAGE**

21373 To ensure applications portability, especially across natural languages, only this function and
21374 those listed in the SEE ALSO section should be used for character classification.

21375 **RATIONALE**

21376 None.

21377 **FUTURE DIRECTIONS**

21378 None.

21379 **SEE ALSO**

21380 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isupper()*,
21381 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
21382 <ctype.h>

21383 **CHANGE HISTORY**

21384 First released in Issue 1. Derived from Issue 1 of the SVID.

21385 **Issue 6**

21386 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21387 **NAME**

21388 isunordered — test if arguments are unordered

21389 **SYNOPSIS**

21390 #include <math.h>

21391 int isunordered(real-floating x, real-floating y);

21392 **DESCRIPTION**

21393 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21394 conflict between the requirements described here and the ISO C standard is unintentional. This
21395 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21396 The *isunordered()* macro shall determine whether its arguments are unordered.

21397 **RETURN VALUE**

21398 Upon successful completion, the *isunordered()* macro shall return 1 if its arguments are
21399 unordered, and 0 otherwise.

21400 If x or y is NaN, 0 shall be returned.

21401 **ERRORS**

21402 No errors are defined.

21403 **EXAMPLES**

21404 None.

21405 **APPLICATION USAGE**

21406 The relational and equality operators support the usual mathematical relationships between
21407 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
21408 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
21409 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
21410 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
21411 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
21412 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
21413 indicates that the argument shall be an expression of **real-floating** type.

21414 **RATIONALE**

21415 None.

21416 **FUTURE DIRECTIONS**

21417 None.

21418 **SEE ALSO**

21419 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*, the Base Definitions volume of
21420 IEEE Std 1003.1-2001, <math.h>

21421 **CHANGE HISTORY**

21422 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21423 **NAME**

21424 isupper — test for an uppercase letter

21425 **SYNOPSIS**

21426 #include <ctype.h>

21427 int isupper(int c);

21428 **DESCRIPTION**

21429 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21430 conflict between the requirements described here and the ISO C standard is unintentional. This
21431 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21432 The *isupper()* function shall test whether *c* is a character of class **upper** in the program's current
21433 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21434 The *c* argument is an **int**, the value of which the application shall ensure is a character
21435 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
21436 any other value, the behavior is undefined.

21437 **RETURN VALUE**21438 The *isupper()* function shall return non-zero if *c* is an uppercase letter; otherwise, it shall return 0.21439 **ERRORS**

21440 No errors are defined.

21441 **EXAMPLES**

21442 None.

21443 **APPLICATION USAGE**

21444 To ensure applications portability, especially across natural languages, only this function and
21445 those listed in the SEE ALSO section should be used for character classification.

21446 **RATIONALE**

21447 None.

21448 **FUTURE DIRECTIONS**

21449 None.

21450 **SEE ALSO**

21451 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isxdigit()*,
21452 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

21453 **CHANGE HISTORY**

21454 First released in Issue 1. Derived from Issue 1 of the SVID.

21455 **Issue 6**

21456 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21457 **NAME**

21458 iswalnum — test for an alphanumeric wide-character code

21459 **SYNOPSIS**

21460 #include <wctype.h>

21461 int iswalnum(wint_t wc);

21462 **DESCRIPTION**

21463 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
21464 conflict between the requirements described here and the ISO C standard is unintentional. This
21465 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21466 The *iswalnum()* function shall test whether *wc* is a wide-character code representing a character
21467 of class **alpha** or **digit** in the program's current locale; see the Base Definitions volume of
21468 IEEE Std 1003.1-2001, Chapter 7, Locale.

21469 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21470 code corresponding to a valid character in the current locale, or equal to the value of the macro
21471 WEOF. If the argument has any other value, the behavior is undefined.

21472 **RETURN VALUE**

21473 The *iswalnum()* function shall return non-zero if *wc* is an alphanumeric wide-character code;
21474 otherwise, it shall return 0.

21475 **ERRORS**

21476 No errors are defined.

21477 **EXAMPLES**

21478 None.

21479 **APPLICATION USAGE**

21480 To ensure applications portability, especially across natural languages, only this function and
21481 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21482 **RATIONALE**

21483 None.

21484 **FUTURE DIRECTIONS**

21485 None.

21486 **SEE ALSO**

21487 *iswalphabet()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
21488 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
21489 IEEE Std 1003.1-2001, Chapter 7, Locale, <stdio.h>, <wchar.h>, <wctype.h>

21490 **CHANGE HISTORY**

21491 First released as a World-wide Portability Interface in Issue 4.

21492 **Issue 5**

21493 The following change has been made in this issue for alignment with
21494 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21495 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21496 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21497 **Issue 6**

21498 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21499 **NAME**

21500 iswalpha — test for an alphabetic wide-character code

21501 **SYNOPSIS**

21502 #include <wctype.h>

21503 int iswalpha(wint_t wc);

21504 **DESCRIPTION**

21505 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 21506 conflict between the requirements described here and the ISO C standard is unintentional. This
 21507 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21508 The *iswalpha()* function shall test whether *wc* is a wide-character code representing a character of
 21509 class **alpha** in the program's current locale; see the Base Definitions volume of
 21510 IEEE Std 1003.1-2001, Chapter 7, Locale.

21511 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 21512 code corresponding to a valid character in the current locale, or equal to the value of the macro
 21513 WEOF. If the argument has any other value, the behavior is undefined.

21514 **RETURN VALUE**

21515 The *iswalpha()* function shall return non-zero if *wc* is an alphabetic wide-character code;
 21516 otherwise, it shall return 0.

21517 **ERRORS**

21518 No errors are defined.

21519 **EXAMPLES**

21520 None.

21521 **APPLICATION USAGE**

21522 To ensure applications portability, especially across natural languages, only this function and
 21523 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21524 **RATIONALE**

21525 None.

21526 **FUTURE DIRECTIONS**

21527 None.

21528 **SEE ALSO**

21529 *iswalnum()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
 21530 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
 21531 IEEE Std 1003.1-2001, Chapter 7, Locale, <stdio.h>, <wchar.h>, <wctype.h>

21532 **CHANGE HISTORY**

21533 First released in Issue 4.

21534 **Issue 5**

21535 The following change has been made in this issue for alignment with
 21536 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21537 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 21538 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21539 **Issue 6**

21540 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21541 **NAME**

21542 iswblank — test for a blank wide-character code

21543 **SYNOPSIS**

21544 #include <wctype.h>

21545 int iswblank(wint_t wc);

21546 **DESCRIPTION**

21547 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21548 conflict between the requirements described here and the ISO C standard is unintentional. This
21549 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21550 The *iswblank()* function shall test whether *wc* is a wide-character code representing a character of
21551 class **blank** in the program's current locale; see the Base Definitions volume of
21552 IEEE Std 1003.1-2001, Chapter 7, Locale.

21553 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21554 code corresponding to a valid character in the current locale, or equal to the value of the macro
21555 WEOF. If the argument has any other value, the behavior is undefined.

21556 **RETURN VALUE**

21557 The *iswblank()* function shall return non-zero if *wc* is a blank wide-character code; otherwise, it
21558 shall return 0.

21559 **ERRORS**

21560 No errors are defined.

21561 **EXAMPLES**

21562 None.

21563 **APPLICATION USAGE**

21564 To ensure applications portability, especially across natural languages, only this function and
21565 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21566 **RATIONALE**

21567 None.

21568 **FUTURE DIRECTIONS**

21569 None.

21570 **SEE ALSO**

21571 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
21572 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
21573 IEEE Std 1003.1-2001, Chapter 7, Locale, <stdio.h>, <wchar.h>, <wctype.h>

21574 **CHANGE HISTORY**

21575 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21576 **NAME**

21577 iswcntrl — test for a control wide-character code

21578 **SYNOPSIS**

21579 #include <wctype.h>

21580 int iswcntrl(wint_t wc);

21581 **DESCRIPTION**

21582 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21583 conflict between the requirements described here and the ISO C standard is unintentional. This
21584 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21585 The *iswcntrl()* function shall test whether *wc* is a wide-character code representing a character of
21586 class **cntrl** in the program's current locale; see the Base Definitions volume of
21587 IEEE Std 1003.1-2001, Chapter 7, Locale.

21588 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21589 code corresponding to a valid character in the current locale, or equal to the value of the macro
21590 WEOF. If the argument has any other value, the behavior is undefined.

21591 **RETURN VALUE**

21592 The *iswcntrl()* function shall return non-zero if *wc* is a control wide-character code; otherwise, it
21593 shall return 0.

21594 **ERRORS**

21595 No errors are defined.

21596 **EXAMPLES**

21597 None.

21598 **APPLICATION USAGE**

21599 To ensure applications portability, especially across natural languages, only this function and
21600 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21601 **RATIONALE**

21602 None.

21603 **FUTURE DIRECTIONS**

21604 None.

21605 **SEE ALSO**

21606 *iswalnum()*, *iswalpha()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
21607 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
21608 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21609 **CHANGE HISTORY**

21610 First released in Issue 4.

21611 **Issue 5**

21612 The following change has been made in this issue for alignment with
21613 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21614 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21615 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21616 **Issue 6**

21617 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21618 **NAME**

21619 iswctype — test character for a specified class

21620 **SYNOPSIS**

21621 #include <wctype.h>

21622 int iswctype(wint_t *wc*, wctype_t *charclass*);21623 **DESCRIPTION**

21624 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21625 conflict between the requirements described here and the ISO C standard is unintentional. This
 21626 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21627 The *iswctype()* function shall determine whether the wide-character code *wc* has the character
 21628 class *charclass*, returning true or false. The *iswctype()* function is defined on WEOF and wide-
 21629 character codes corresponding to the valid character encodings in the current locale. If the *wc*
 21630 argument is not in the domain of the function, the result is undefined. If the value of *charclass* is
 21631 invalid (that is, not obtained by a call to *wctype()* or *charclass* is invalidated by a subsequent call
 21632 to *setlocale()* that has affected category *LC_CTYPE*) the result is unspecified.

21633 **RETURN VALUE**

21634 The *iswctype()* function shall return non-zero (true) if and only if *wc* has the property described
 21635 CX by *charclass*. If *charclass* is 0, *iswctype()* shall return 0.

21636 **ERRORS**

21637 No errors are defined.

21638 **EXAMPLES**21639 **Testing for a Valid Character**

```
21640 #include <wctype.h>
21641 ...
21642 int yes_or_no;
21643 wint_t wc;
21644 wctype_t valid_class;
21645 ...
21646 if ((valid_class=wctype("vowel")) == (wctype_t)0)
21647     /* Invalid character class. */
21648     yes_or_no=iswctype(wc,valid_class);
```

21649 **APPLICATION USAGE**

21650 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",
 21651 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard
 21652 character classes. In the table below, the functions in the left column are equivalent to the
 21653 functions in the right column.

21654 iswalnum(<i>wc</i>)	iswctype(<i>wc</i> , wctype("alnum"))
21655 iswalpha(<i>wc</i>)	iswctype(<i>wc</i> , wctype("alpha"))
21656 iswblank(<i>wc</i>)	iswctype(<i>wc</i> , wctype("blank"))
21657 iswcntrl(<i>wc</i>)	iswctype(<i>wc</i> , wctype("cntrl"))
21658 iswdigit(<i>wc</i>)	iswctype(<i>wc</i> , wctype("digit"))
21659 iswgraph(<i>wc</i>)	iswctype(<i>wc</i> , wctype("graph"))
21660 iswlower(<i>wc</i>)	iswctype(<i>wc</i> , wctype("lower"))
21661 iswprint(<i>wc</i>)	iswctype(<i>wc</i> , wctype("print"))
21662 iswpunct(<i>wc</i>)	iswctype(<i>wc</i> , wctype("punct"))
21663 iswspace(<i>wc</i>)	iswctype(<i>wc</i> , wctype("space"))

```

21664      iswupper(wc)      iswctype(wc, wctype("upper"))
21665      iswxdigit(wc)      iswctype(wc, wctype("xdigit"))

```

21666 RATIONALE

21667 None.

21668 FUTURE DIRECTIONS

21669 None.

21670 SEE ALSO

21671 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
 21672 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *wctype()*, the Base Definitions volume of
 21673 IEEE Std 1003.1-2001, **<wchar.h>**, **<wctype.h>**

21674 CHANGE HISTORY

21675 First released as World-wide Portability Interfaces in Issue 4.

21676 Issue 5

21677 The following change has been made in this issue for alignment with
 21678 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21679 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 21680 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

21681 Issue 6

21682 The behavior of $n=0$ is now described.

21683 An example is added.

21684 A new function, *iswblank()*, is added to the list in the APPLICATION USAGE.

21685 **NAME**

21686 iswdigit — test for a decimal digit wide-character code

21687 **SYNOPSIS**

21688 #include <wctype.h>

21689 int iswdigit(wint_t wc);

21690 **DESCRIPTION**

21691 cx The functionality described on this reference page is aligned with the ISO C standard. Any
21692 conflict between the requirements described here and the ISO C standard is unintentional. This
21693 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21694 The *iswdigit()* function shall test whether *wc* is a wide-character code representing a character of
21695 class **digit** in the program's current locale; see the Base Definitions volume of
21696 IEEE Std 1003.1-2001, Chapter 7, Locale.

21697 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
21698 code corresponding to a valid character in the current locale, or equal to the value of the macro
21699 WEOF. If the argument has any other value, the behavior is undefined.

21700 **RETURN VALUE**

21701 The *iswdigit()* function shall return non-zero if *wc* is a decimal digit wide-character code;
21702 otherwise, it shall return 0.

21703 **ERRORS**

21704 No errors are defined.

21705 **EXAMPLES**

21706 None.

21707 **APPLICATION USAGE**

21708 To ensure applications portability, especially across natural languages, only this function and
21709 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21710 **RATIONALE**

21711 None.

21712 **FUTURE DIRECTIONS**

21713 None.

21714 **SEE ALSO**

21715 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
21716 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
21717 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21718 **CHANGE HISTORY**

21719 First released in Issue 4.

21720 **Issue 5**

21721 The following change has been made in this issue for alignment with
21722 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21723 • The SYNOPSIS has been changed to indicate that this function and associated data types are
21724 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21725 **Issue 6**

21726 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21727 **NAME**

21728 iswgraph — test for a visible wide-character code

21729 **SYNOPSIS**

21730 #include <wctype.h>

21731 int iswgraph(wint_t wc);

21732 **DESCRIPTION**

21733 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 21734 conflict between the requirements described here and the ISO C standard is unintentional. This
 21735 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21736 The *iswgraph()* function shall test whether *wc* is a wide-character code representing a character
 21737 of class **graph** in the program's current locale; see the Base Definitions volume of
 21738 IEEE Std 1003.1-2001, Chapter 7, Locale.

21739 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 21740 code corresponding to a valid character in the current locale, or equal to the value of the macro
 21741 WEOF. If the argument has any other value, the behavior is undefined.

21742 **RETURN VALUE**

21743 The *iswgraph()* function shall return non-zero if *wc* is a wide-character code with a visible
 21744 representation; otherwise, it shall return 0.

21745 **ERRORS**

21746 No errors are defined.

21747 **EXAMPLES**

21748 None.

21749 **APPLICATION USAGE**

21750 To ensure applications portability, especially across natural languages, only this function and
 21751 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21752 **RATIONALE**

21753 None.

21754 **FUTURE DIRECTIONS**

21755 None.

21756 **SEE ALSO**

21757 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswlower()*, *iswprint()*, *iswpunct()*,
 21758 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
 21759 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21760 **CHANGE HISTORY**

21761 First released in Issue 4.

21762 **Issue 5**

21763 The following change has been made in this issue for alignment with
 21764 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21765 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 21766 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21767 **Issue 6**

21768 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21769 **NAME**

21770 iswlower — test for a lowercase letter wide-character code

21771 **SYNOPSIS**

21772 #include <wctype.h>

21773 int iswlower(wint_t wc);

21774 **DESCRIPTION**

21775 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 21776 conflict between the requirements described here and the ISO C standard is unintentional. This
 21777 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21778 The *iswlower()* function shall test whether *wc* is a wide-character code representing a character
 21779 of class **lower** in the program's current locale; see the Base Definitions volume of
 21780 IEEE Std 1003.1-2001, Chapter 7, Locale.

21781 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 21782 code corresponding to a valid character in the current locale, or equal to the value of the macro
 21783 WEOF. If the argument has any other value, the behavior is undefined.

21784 **RETURN VALUE**

21785 The *iswlower()* function shall return non-zero if *wc* is a lowercase letter wide-character code;
 21786 otherwise, it shall return 0.

21787 **ERRORS**

21788 No errors are defined.

21789 **EXAMPLES**

21790 None.

21791 **APPLICATION USAGE**

21792 To ensure applications portability, especially across natural languages, only this function and
 21793 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21794 **RATIONALE**

21795 None.

21796 **FUTURE DIRECTIONS**

21797 None.

21798 **SEE ALSO**

21799 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswprint()*, *iswpunct()*,
 21800 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
 21801 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21802 **CHANGE HISTORY**

21803 First released in Issue 4.

21804 **Issue 5**

21805 The following change has been made in this issue for alignment with
 21806 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21807 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 21808 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21809 **Issue 6**

21810 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21811 **NAME**

21812 iswprint — test for a printable wide-character code

21813 **SYNOPSIS**

21814 #include <wctype.h>

21815 int iswprint(wint_t wc);

21816 **DESCRIPTION**

21817 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 21818 conflict between the requirements described here and the ISO C standard is unintentional. This
 21819 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21820 The *iswprint()* function shall test whether *wc* is a wide-character code representing a character of
 21821 class **print** in the program's current locale; see the Base Definitions volume of
 21822 IEEE Std 1003.1-2001, Chapter 7, Locale.

21823 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 21824 code corresponding to a valid character in the current locale, or equal to the value of the macro
 21825 WEOF. If the argument has any other value, the behavior is undefined.

21826 **RETURN VALUE**

21827 The *iswprint()* function shall return non-zero if *wc* is a printable wide-character code; otherwise,
 21828 it shall return 0.

21829 **ERRORS**

21830 No errors are defined.

21831 **EXAMPLES**

21832 None.

21833 **APPLICATION USAGE**

21834 To ensure applications portability, especially across natural languages, only this function and
 21835 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21836 **RATIONALE**

21837 None.

21838 **FUTURE DIRECTIONS**

21839 None.

21840 **SEE ALSO**

21841 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswpunct()*,
 21842 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
 21843 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21844 **CHANGE HISTORY**

21845 First released in Issue 4.

21846 **Issue 5**

21847 The following change has been made in this issue for alignment with
 21848 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21849 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 21850 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21851 **Issue 6**

21852 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21853 **NAME**

21854 iswpunct — test for a punctuation wide-character code

21855 **SYNOPSIS**

21856 #include <wctype.h>

21857 int iswpunct(wint_t wc);

21858 **DESCRIPTION**

21859 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 21860 conflict between the requirements described here and the ISO C standard is unintentional. This
 21861 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21862 The *iswpunct()* function shall test whether *wc* is a wide-character code representing a character
 21863 of class **punct** in the program's current locale; see the Base Definitions volume of
 21864 IEEE Std 1003.1-2001, Chapter 7, Locale.

21865 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 21866 code corresponding to a valid character in the current locale, or equal to the value of the macro
 21867 WEOF. If the argument has any other value, the behavior is undefined.

21868 **RETURN VALUE**

21869 The *iswpunct()* function shall return non-zero if *wc* is a punctuation wide-character code;
 21870 otherwise, it shall return 0.

21871 **ERRORS**

21872 No errors are defined.

21873 **EXAMPLES**

21874 None.

21875 **APPLICATION USAGE**

21876 To ensure applications portability, especially across natural languages, only this function and
 21877 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21878 **RATIONALE**

21879 None.

21880 **FUTURE DIRECTIONS**

21881 None.

21882 **SEE ALSO**

21883 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
 21884 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
 21885 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21886 **CHANGE HISTORY**

21887 First released in Issue 4.

21888 **Issue 5**

21889 The following change has been made in this issue for alignment with
 21890 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21891 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 21892 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21893 **Issue 6**

21894 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21895 **NAME**

21896 iswspace — test for a white-space wide-character code

21897 **SYNOPSIS**

21898 #include <wctype.h>

21899 int iswspace(wint_t wc);

21900 **DESCRIPTION**

21901 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 21902 conflict between the requirements described here and the ISO C standard is unintentional. This
 21903 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21904 The *iswspace()* function shall test whether *wc* is a wide-character code representing a character of
 21905 class **space** in the program's current locale; see the Base Definitions volume of
 21906 IEEE Std 1003.1-2001, Chapter 7, Locale.

21907 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 21908 code corresponding to a valid character in the current locale, or equal to the value of the macro
 21909 WEOF. If the argument has any other value, the behavior is undefined.

21910 **RETURN VALUE**

21911 The *iswspace()* function shall return non-zero if *wc* is a white-space wide-character code;
 21912 otherwise, it shall return 0.

21913 **ERRORS**

21914 No errors are defined.

21915 **EXAMPLES**

21916 None.

21917 **APPLICATION USAGE**

21918 To ensure applications portability, especially across natural languages, only this function and
 21919 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21920 **RATIONALE**

21921 None.

21922 **FUTURE DIRECTIONS**

21923 None.

21924 **SEE ALSO**

21925 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
 21926 *iswpunct()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
 21927 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21928 **CHANGE HISTORY**

21929 First released in Issue 4.

21930 **Issue 5**

21931 The following change has been made in this issue for alignment with
 21932 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21933 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 21934 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21935 **Issue 6**

21936 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21937 **NAME**

21938 iswupper — test for an uppercase letter wide-character code

21939 **SYNOPSIS**

21940 #include <wctype.h>

21941 int iswupper(wint_t wc);

21942 **DESCRIPTION**

21943 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 21944 conflict between the requirements described here and the ISO C standard is unintentional. This
 21945 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21946 The *iswupper()* function shall test whether *wc* is a wide-character code representing a character
 21947 of class **upper** in the program's current locale; see the Base Definitions volume of
 21948 IEEE Std 1003.1-2001, Chapter 7, Locale.

21949 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 21950 code corresponding to a valid character in the current locale, or equal to the value of the macro
 21951 WEOF. If the argument has any other value, the behavior is undefined.

21952 **RETURN VALUE**

21953 The *iswupper()* function shall return non-zero if *wc* is an uppercase letter wide-character code;
 21954 otherwise, it shall return 0.

21955 **ERRORS**

21956 No errors are defined.

21957 **EXAMPLES**

21958 None.

21959 **APPLICATION USAGE**

21960 To ensure applications portability, especially across natural languages, only this function and
 21961 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21962 **RATIONALE**

21963 None.

21964 **FUTURE DIRECTIONS**

21965 None.

21966 **SEE ALSO**

21967 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
 21968 *iswpunct()*, *iswspace()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of
 21969 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21970 **CHANGE HISTORY**

21971 First released in Issue 4.

21972 **Issue 5**

21973 The following change has been made in this issue for alignment with
 21974 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21975 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 21976 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21977 **Issue 6**

21978 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21979 **NAME**

21980 iswxdigit — test for a hexadecimal digit wide-character code

21981 **SYNOPSIS**

21982 #include <wctype.h>

21983 int iswxdigit(wint_t wc);

21984 **DESCRIPTION**

21985 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 21986 conflict between the requirements described here and the ISO C standard is unintentional. This
 21987 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21988 The *iswxdigit()* function shall test whether *wc* is a wide-character code representing a character
 21989 of class **xdigit** in the program's current locale; see the Base Definitions volume of
 21990 IEEE Std 1003.1-2001, Chapter 7, Locale.

21991 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 21992 code corresponding to a valid character in the current locale, or equal to the value of the macro
 21993 WEOF. If the argument has any other value, the behavior is undefined.

21994 **RETURN VALUE**

21995 The *iswxdigit()* function shall return non-zero if *wc* is a hexadecimal digit wide-character code;
 21996 otherwise, it shall return 0.

21997 **ERRORS**

21998 No errors are defined.

21999 **EXAMPLES**

22000 None.

22001 **APPLICATION USAGE**

22002 To ensure applications portability, especially across natural languages, only this function and
 22003 those listed in the SEE ALSO section should be used for classification of wide-character codes.

22004 **RATIONALE**

22005 None.

22006 **FUTURE DIRECTIONS**

22007 None.

22008 **SEE ALSO**

22009 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
 22010 *iswpunct()*, *iswspace()*, *iswupper()*, *setlocale()*, the Base Definitions volume of
 22011 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

22012 **CHANGE HISTORY**

22013 First released in Issue 4.

22014 **Issue 5**

22015 The following change has been made in this issue for alignment with
 22016 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22017 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 22018 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

22019 **Issue 6**

22020 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22021 **NAME**

22022 isxdigit — test for a hexadecimal digit

22023 **SYNOPSIS**

22024 #include <ctype.h>

22025 int isxdigit(int c);

22026 **DESCRIPTION**

22027 cx The functionality described on this reference page is aligned with the ISO C standard. Any
22028 conflict between the requirements described here and the ISO C standard is unintentional. This
22029 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22030 The *isxdigit()* function shall test whether *c* is a character of class **xdigit** in the program's current
22031 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

22032 The *c* argument is an **int**, the value of which the application shall ensure is a character
22033 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
22034 any other value, the behavior is undefined.

22035 **RETURN VALUE**

22036 The *isxdigit()* function shall return non-zero if *c* is a hexadecimal digit; otherwise, it shall return
22037 0.

22038 **ERRORS**

22039 No errors are defined.

22040 **EXAMPLES**

22041 None.

22042 **APPLICATION USAGE**

22043 To ensure applications portability, especially across natural languages, only this function and
22044 those listed in the SEE ALSO section should be used for character classification.

22045 **RATIONALE**

22046 None.

22047 **FUTURE DIRECTIONS**

22048 None.

22049 **SEE ALSO**

22050 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
22051 the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

22052 **CHANGE HISTORY**

22053 First released in Issue 1. Derived from Issue 1 of the SVID.

22054 **Issue 6**

22055 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22056 **NAME**

22057 j0, j1, jn — Bessel functions of the first kind

22058 **SYNOPSIS**

```

22059 xSI      #include <math.h>

22060          double j0(double x);
22061          double j1(double x);
22062          double jn(int n, double x);
22063

```

22064 **DESCRIPTION**

22065 The *j0()*, *j1()*, and *jn()* functions shall compute Bessel functions of *x* of the first kind of orders 0, 1, and *n*, respectively.

22067 An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

22071 **RETURN VALUE**

22072 Upon successful completion, these functions shall return the relevant Bessel value of *x* of the first kind.

22074 If the *x* argument is too large in magnitude, or the correct result would cause underflow, 0 shall be returned and a range error may occur.

22076 If *x* is NaN, a NaN shall be returned.

22077 **ERRORS**

22078 These functions may fail if:

22079 Range Error The value of *x* was too large in magnitude, or an underflow occurred.

22080 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

22084 No other errors shall occur.

22085 **EXAMPLES**

22086 None.

22087 **APPLICATION USAGE**

22088 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22090 **RATIONALE**

22091 None.

22092 **FUTURE DIRECTIONS**

22093 None.

22094 **SEE ALSO**

22095 *feclearexcept()*, *fetestexcept()*, *isnan()*, *y0()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

22097 CHANGE HISTORY

22098 First released in Issue 1. Derived from Issue 1 of the SVID.

22099 Issue 5

22100 The DESCRIPTION is updated to indicate how an application should check for an error. This
22101 text was previously published in the APPLICATION USAGE section.

22102 Issue 6

22103 The may fail [EDOM] error is removed for the case for NaN.

22104 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
22105 with the ISO / IEC 9899:1999 standard.

22106 **NAME**

22107 jrand48 — generate a uniformly distributed pseudo-random long signed integer

22108 **SYNOPSIS**

22109 xSI #include <stdlib.h>

22110 long jrand48(unsigned short xsubi[3]);

22111

22112 **DESCRIPTION**22113 Refer to *drand48()*.

22114 **NAME**

22115 kill — send a signal to a process or a group of processes

22116 **SYNOPSIS**22117 CX `#include <signal.h>`22118 `int kill(pid_t pid, int sig);`

22119

22120 **DESCRIPTION**

22121 The `kill()` function shall send a signal to a process or a group of processes specified by *pid*. The
 22122 signal to be sent is specified by *sig* and is either one from the list given in **<signal.h>** or 0. If *sig* is
 22123 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can
 22124 be used to check the validity of *pid*.

22125 For a process to have permission to send a signal to a process designated by *pid*, unless the
 22126 sending process has appropriate privileges, the real or effective user ID of the sending process
 22127 shall match the real or saved set-user-ID of the receiving process.

22128 If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

22129 If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes)
 22130 whose process group ID is equal to the process group ID of the sender, and for which the
 22131 process has permission to send a signal.

22132 If *pid* is -1 , *sig* shall be sent to all processes (excluding an unspecified set of system processes) for
 22133 which the process has permission to send that signal.

22134 If *pid* is negative, but not -1 , *sig* shall be sent to all processes (excluding an unspecified set of
 22135 system processes) whose process group ID is equal to the absolute value of *pid*, and for which
 22136 the process has permission to send a signal.

22137 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for
 22138 the calling thread and if no other thread has *sig* unblocked or is waiting in a `sigwait()` function
 22139 for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending
 22140 thread before `kill()` returns.

22141 The user ID tests described above shall not be applied when sending SIGCONT to a process that
 22142 is a member of the same session as the sending process.

22143 An implementation that provides extended security controls may impose further
 22144 implementation-defined restrictions on the sending of signals, including the null signal. In
 22145 particular, the system may deny the existence of some or all of the processes specified by *pid*.

22146 The `kill()` function is successful if the process has permission to send *sig* to any of the processes
 22147 specified by *pid*. If `kill()` fails, no signal shall be sent.

22148 **RETURN VALUE**

22149 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 22150 indicate the error.

22151 **ERRORS**

22152 The `kill()` function shall fail if:

22153	[EINVAL]	The value of the <i>sig</i> argument is an invalid or unsupported signal number.
22154	[EPERM]	The process does not have permission to send the signal to any receiving
22155		process.
22156	[ESRCH]	No process or process group can be found corresponding to that specified by
22157		<i>pid</i> .

22158 **EXAMPLES**

22159 None.

22160 **APPLICATION USAGE**

22161 None.

22162 **RATIONALE**

22163 The semantics for permission checking for *kill()* differed between System V and most other
22164 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of
22165 IEEE Std 1003.1-2001 agree with System V. Specifically, a set-user-ID process cannot protect
22166 itself against signals (or at least not against SIGKILL) unless it changes its real user ID. This
22167 choice allows the user who starts an application to send it signals even if it changes its effective
22168 user ID. The other semantics give more power to an application that wants to protect itself from
22169 the user who ran it.

22170 Some implementations provide semantic extensions to the *kill()* function when the absolute
22171 value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a
22172 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not
22173 included in this volume of IEEE Std 1003.1-2001, although a conforming implementation could
22174 provide such an extension.

22175 The implementation-defined processes to which a signal cannot be sent may include the
22176 scheduler or *init*.

22177 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the
22178 calling process and that signal is not blocked, that signal would be delivered before *kill()*
22179 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.
22180 However, historical implementations that provide only the *signal()* function make only the
22181 weaker guarantee in this volume of IEEE Std 1003.1-2001, because they only deliver one signal
22182 each time a process enters the kernel. Modifications to such implementations to support the
22183 *sigaction()* function generally require entry to the kernel following return from a signal-catching
22184 function, in order to restore the signal mask. Such modifications have the effect of satisfying the
22185 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.
22186 The developers of this volume of IEEE Std 1003.1-2001 considered making the stronger
22187 requirement except when *signal()* is used, but felt this would be unnecessarily complex.
22188 Implementors are encouraged to meet the stronger requirement whenever possible. In practice,
22189 the weaker requirement is the same, except in the rare case when two signals arrive during a
22190 very short window. This reasoning also applies to a similar requirement for *sigprocmask()*.

22191 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID
22192 security checks. This allows a job control shell to continue a job even if processes in the job have
22193 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of
22194 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any
22195 process in the same session regardless of user ID security checks. This is less restrictive than BSD
22196 in the sense that ancestor processes (in the same session) can now be the recipient. It is more
22197 restrictive than BSD in the sense that descendant processes that form new sessions are now
22198 subject to the user ID checks. A similar relaxation of security is not necessary for the other job
22199 control signals since those signals are typically sent by the terminal driver in recognition of
22200 special characters being typed; the terminal driver bypasses all security checks.

22201 In secure implementations, a process may be restricted from sending a signal to a process having
22202 a different security label. In order to prevent the existence or nonexistence of a process from
22203 being used as a covert channel, such processes should appear nonexistent to the sender; that is,
22204 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

Existing implementations vary on the result of a *kill()* with *pid* indicating an inactive process (a terminated process that has not been waited for by its parent). Some indicate success on such a call (subject to permission checking), while others give an error of [ESRCH]. Since the definition of process lifetime in this volume of IEEE Std 1003.1-2001 covers inactive processes, the [ESRCH] error as described is inappropriate in this case. In particular, this means that an application cannot have a parent process check for termination of a particular child with *kill()*. (Usually this is done with the null signal; this can be done reliably with *waitpid()*.)

There is some belief that the name *kill()* is misleading, since the function is not always intended to cause process termination. However, the name is common to all historical implementations, and any change would be in conflict with the goal of minimal changes to existing application code.

22216 FUTURE DIRECTIONS

22217 None.

22218 SEE ALSO

22219 *getpid()*, *raise()*, *setsid()*, *sigaction()*, *sigqueue()*, the Base Definitions volume of
22220 IEEE Std 1003.1-2001, **<signal.h>**, **<sys/types.h>**

22221 CHANGE HISTORY

22222 First released in Issue 1. Derived from Issue 1 of the SVID.

22223 Issue 5

22224 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

22225 Issue 6

22226 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

22227 The following new requirements on POSIX implementations derive from alignment with the
22228 Single UNIX Specification:

- 22229 • In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-
22230 ID of the calling process is checked in place of its effective user ID. This is a FIPS
22231 requirement.
- 22232 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
22233 required for conforming implementations of previous POSIX specifications, it was not
22234 required for UNIX applications.
- 22235 • The behavior when *pid* is *-1* is now specified. It was previously explicitly unspecified in the
22236 POSIX.1-1988 standard.

22237 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22238 **NAME**

22239 killpg — send a signal to a process group

22240 **SYNOPSIS**

22241 xSI #include <signal.h>

22242 int killpg(pid_t pgrp, int sig);

22243

22244 **DESCRIPTION**22245 The *killpg()* function shall send the signal specified by *sig* to the process group specified by *pgrp*.22246 If *pgrp* is greater than 1, *killpg(pgrp, sig)* shall be equivalent to *kill(-pgrp, sig)*. If *pgrp* is less than or
22247 equal to 1, the behavior of *killpg()* is undefined.22248 **RETURN VALUE**22249 Refer to *kill()*.22250 **ERRORS**22251 Refer to *kill()*.22252 **EXAMPLES**

22253 None.

22254 **APPLICATION USAGE**

22255 None.

22256 **RATIONALE**

22257 None.

22258 **FUTURE DIRECTIONS**

22259 None.

22260 **SEE ALSO**22261 *getpgid()*, *getpid()*, *kill()*, *raise()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>22262 **CHANGE HISTORY**

22263 First released in Issue 4, Version 2.

22264 **Issue 5**

22265 Moved from X/OPEN UNIX extension to BASE.

22266 NAME

22267 l64a — convert a 32-bit integer to a radix-64 ASCII string

22268 SYNOPSIS

22269 xSI #include <stdlib.h>

22270 char *l64a(long value);

22271

22272 DESCRIPTION

22273 Refer to *a64l()*.

22274 **NAME**

22275 labs, llabs — return a long integer absolute value

22276 **SYNOPSIS**

22277 #include <stdlib.h>

22278 long labs(long i);

22279 long long llabs(long long i);

22280 **DESCRIPTION**

22281 cx The functionality described on this reference page is aligned with the ISO C standard. Any
22282 conflict between the requirements described here and the ISO C standard is unintentional. This
22283 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22284 The *labs()* function shall compute the absolute value of the **long** integer operand *i*. The *llabs()*
22285 function shall compute the absolute value of the **long long** integer operand *i*. If the result cannot
22286 be represented, the behavior is undefined.

22287 **RETURN VALUE**

22288 The *labs()* function shall return the absolute value of the **long** integer operand. The *labs()*
22289 function shall return the absolute value of the **long long** integer operand.

22290 **ERRORS**

22291 No errors are defined.

22292 **EXAMPLES**

22293 None.

22294 **APPLICATION USAGE**

22295 None.

22296 **RATIONALE**

22297 None.

22298 **FUTURE DIRECTIONS**

22299 None.

22300 **SEE ALSO**

22301 *abs()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

22302 **CHANGE HISTORY**

22303 First released in Issue 4. Derived from the ISO C standard.

22304 **Issue 6**

22305 The *llabs()* function is added for alignment with the ISO / IEC 9899:1999 standard.

22306 **NAME**

22307 lchown — change the owner and group of a symbolic link

22308 **SYNOPSIS**22309 xsi `#include <unistd.h>`22310 `int lchown(const char *path, uid_t owner, gid_t group);`

22311

22312 **DESCRIPTION**

22313 The *lchown()* function shall be equivalent to *chown()*, except in the case where the named file is a
 22314 symbolic link. In this case, *lchown()* shall change the ownership of the symbolic link file itself,
 22315 while *chown()* changes the ownership of the file or directory to which the symbolic link refers.

22316 **RETURN VALUE**

22317 Upon successful completion, *lchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to
 22318 indicate an error.

22319 **ERRORS**22320 The *lchown()* function shall fail if:22321 [EACCES] Search permission is denied on a component of the path prefix of *path*.

22322 [EINVAL] The owner or group ID is not a value supported by the implementation.

22323 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 22324 argument.

22325 [ENAMETOOLONG]

22326 The length of a pathname exceeds {PATH_MAX} or a pathname component is
 22327 longer than {NAME_MAX}.

22328 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.22329 [ENOTDIR] A component of the path prefix of *path* is not a directory.

22330 [EOPNOTSUPP] The *path* argument names a symbolic link and the implementation does not
 22331 support setting the owner or group of a symbolic link.

22332 [EPERM] The effective user ID does not match the owner of the file and the process
 22333 does not have appropriate privileges.

22334 [EROFS] The file resides on a read-only file system.

22335 The *lchown()* function may fail if:

22336 [EIO] An I/O error occurred while reading or writing to the file system.

22337 [EINTR] A signal was caught during execution of the function.

22338 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 22339 resolution of the *path* argument.

22340 [ENAMETOOLONG]

22341 Pathname resolution of a symbolic link produced an intermediate result
 22342 whose length exceeds {PATH_MAX}.

22343 EXAMPLES**22344 Changing the Current Owner of a File**

22345 The following example shows how to change the ownership of the symbolic link named
22346 **/modules/pass1** to the user ID associated with “jones” and the group ID associated with “cnd”.

22347 The numeric value for the user ID is obtained by using the *getpwnam()* function. The numeric
22348 value for the group ID is obtained by using the *getgrnam()* function.

```
22349 #include <sys/types.h>
22350 #include <unistd.h>
22351 #include <pwd.h>
22352 #include <grp.h>

22353 struct passwd *pwd;
22354 struct group *grp;
22355 char          *path = "/modules/pass1";
22356 ...
22357 pwd = getpwnam("jones");
22358 grp = getgrnam("cnd");
22359 lchown(path, pwd->pw_uid, grp->gr_gid);
```

22360 APPLICATION USAGE

22361 On implementations which support symbolic links as directory entries rather than files, *lchown()*
22362 may fail.

22363 RATIONALE

22364 None.

22365 FUTURE DIRECTIONS

22366 None.

22367 SEE ALSO

22368 *chown()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

22369 CHANGE HISTORY

22370 First released in Issue 4, Version 2.

22371 Issue 5

22372 Moved from X/OPEN UNIX extension to BASE.

22373 Issue 6

22374 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
22375 [ELOOP] error condition is added.

22376 The Open Group Base Resolution bwg2001-013 is applied, adding wording to the
22377 APPLICATION USAGE.

22378 NAME

22379 lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

22380 SYNOPSIS

22381 xSI #include <stdlib.h>

22382 void lcong48(unsigned short param[7]);

22383

22384 DESCRIPTION

22385 Refer to *drand48()*.

22386 **NAME**

22387 ldexp, ldexpf, ldexpl — load exponent of a floating-point number

22388 **SYNOPSIS**

22389 #include <math.h>

22390 double ldexp(double x, int exp);

22391 float ldexpf(float x, int exp);

22392 long double ldexpl(long double x, int exp);

22393 **DESCRIPTION**

22394 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22395 conflict between the requirements described here and the ISO C standard is unintentional. This
 22396 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22397 These functions shall compute the quantity $x * 2^{exp}$.

22398 An application wishing to check for error situations should set *errno* to zero and call
 22399 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 22400 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 22401 zero, an error has occurred.

22402 **RETURN VALUE**22403 Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power
22404 *exp*.

22405 If these functions would cause overflow, a range error shall occur and *ldexp*(), *ldexpf*(), and
 22406 *ldexpl*() shall return $\pm\text{HUGE\_VAL}$, $\pm\text{HUGE\_VALF}$, and $\pm\text{HUGE\_VALL}$ (according to the sign of
 22407 *x*), respectively.

22408 If the correct value would cause underflow, and is not representable, a range error may occur,
 22409 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

22410 MX If *x* is NaN, a NaN shall be returned.22411 If *x* is ± 0 or $\pm\text{Inf}$, *x* shall be returned.22412 If *exp* is 0, *x* shall be returned.

22413 If the correct value would cause underflow, and is representable, a range error may occur and
 22414 the correct value shall be returned.

22415 **ERRORS**

22416 These functions shall fail if:

22417 Range Error The result overflows.

22418 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 22419 then *errno* shall be set to [ERANGE]. If the integer expression
 22420 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 22421 floating-point exception shall be raised.

22422 These functions may fail if:

22423 Range Error The result underflows.

22424 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 22425 then *errno* shall be set to [ERANGE]. If the integer expression
 22426 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 22427 floating-point exception shall be raised.

22428 **EXAMPLES**

22429 None.

22430 **APPLICATION USAGE**

22431 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
22432 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22433 **RATIONALE**

22434 None.

22435 **FUTURE DIRECTIONS**

22436 None.

22437 **SEE ALSO**

22438 *feclearexcept()*, *fetestexcept()*, *frexp()*, *isnan()*, the Base Definitions volume of
22439 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
22440 <math.h>

22441 **CHANGE HISTORY**

22442 First released in Issue 1. Derived from Issue 1 of the SVID.

22443 **Issue 5**

22444 The DESCRIPTION is updated to indicate how an application should check for an error. This
22445 text was previously published in the APPLICATION USAGE section.

22446 **Issue 6**

22447 The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999
22448 standard.

22449 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
22450 revised to align with the ISO/IEC 9899:1999 standard.

22451 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
22452 marked.

22453 **NAME**

22454 ldiv, lldiv — compute quotient and remainder of a long division

22455 **SYNOPSIS**

22456 #include <stdlib.h>

22457 ldiv_t ldiv(long *numer*, long *denom*);22458 lldiv_t lldiv(long long *numer*, long long *denom*);22459 **DESCRIPTION**

22460 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22461 conflict between the requirements described here and the ISO C standard is unintentional. This
 22462 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22463 These functions shall compute the quotient and remainder of the division of the numerator
 22464 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**
 22465 integer (for the *ldiv()* function) or **long long** integer (for the *lldiv()* function) of lesser magnitude
 22466 that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is
 22467 undefined; otherwise, *quot * denom + rem* shall equal *numer*.

22468 **RETURN VALUE**

22469 The *ldiv()* function shall return a structure of type **ldiv_t**, comprising both the quotient and the
 22470 remainder. The structure shall include the following members, in any order:

22471 long quot; /* Quotient */
 22472 long rem; /* Remainder */

22473 The *lldiv()* function shall return a structure of type **lldiv_t**, comprising both the quotient and the
 22474 remainder. The structure shall include the following members, in any order:

22475 long long quot; /* Quotient */
 22476 long long rem; /* Remainder */

22477 **ERRORS**

22478 No errors are defined.

22479 **EXAMPLES**

22480 None.

22481 **APPLICATION USAGE**

22482 None.

22483 **RATIONALE**

22484 None.

22485 **FUTURE DIRECTIONS**

22486 None.

22487 **SEE ALSO**22488 *div()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>22489 **CHANGE HISTORY**

22490 First released in Issue 4. Derived from the ISO C standard.

22491 **Issue 6**22492 The *lldiv()* function is added for alignment with the ISO / IEC 9899:1999 standard.

22493 NAME

22494 lfind — find entry in a linear search table

22495 SYNOPSIS

22496 xSI #include <search.h>

```
22497       void *lfind(const void *key, const void *base, size_t *nelp,  
22498                   size_t width, int (*compar)(const void *, const void *));
```

22499

22500 DESCRIPTION

22501 Refer to *lsearch()*.

22502 **NAME**

22503 lgamma, lgammaf, lgammal — log gamma function

22504 **SYNOPSIS**

22505 #include <math.h>

22506 double lgamma(double x);

22507 float lgammaf(float x);

22508 long double lgammal(long double x);

22509 xsi extern int signgam;

22510

22511 **DESCRIPTION**

22512 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 22513 conflict between the requirements described here and the ISO C standard is unintentional. This
 22514 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22515 These functions shall compute $\log_e |\Gamma(x)|$ where $\Gamma(x)$ is defined as $\int_0^{\infty} e^{-t} t^{x-1} dt$. The argument x
 22516 need not be a non-positive integer ($\Gamma(x)$ is defined over the reals, except the non-positive
 22517 integers).
 22518

22519 xsi The sign of $\Gamma(x)$ is returned in the external integer *signgam*.

22520 cx These functions need not be reentrant. A function that is not required to be reentrant is not
 22521 required to be thread-safe.

22522 An application wishing to check for error situations should set *errno* to zero and call
 22523 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 22524 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 22525 zero, an error has occurred.

22526 **RETURN VALUE**22527 Upon successful completion, these functions shall return the logarithmic gamma of x .

22528 If x is a non-positive integer, a pole error shall occur and *lgamma()*, *lgammaf()*, and *lgammal()*
 22529 shall return +HUGE_VAL, +HUGE_VALF, and +HUGE_VALL, respectively.

22530 If the correct value would cause overflow, a range error shall occur and *lgamma()*, *lgammaf()*,
 22531 and *lgammal()* shall return $\pm$ HUGE_VAL, $\pm$ HUGE_VALF, and $\pm$ HUGE_VALL (having the same
 22532 sign as the correct value), respectively.

22533 mx If x is NaN, a NaN shall be returned.22534 If x is 1 or 2, +0 shall be returned.22535 If x is $\pm$ Inf, +Inf shall be returned.22536 **ERRORS**

22537 These functions shall fail if:

22538 Pole Error The x argument is a negative integer or zero.

22539 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 22540 then *errno* shall be set to [ERANGE]. If the integer expression
 22541 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
 22542 zero floating-point exception shall be raised.

22543 Range Error The result overflows.

22544 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,
22545 then *errno* shall be set to `[ERANGE]`. If the integer expression
22546 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the overflow
22547 floating-point exception shall be raised.

22548 EXAMPLES

22549 None.

22550 APPLICATION USAGE

22551 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
22552 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

22553 RATIONALE

22554 None.

22555 FUTURE DIRECTIONS

22556 None.

22557 SEE ALSO

22558 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
22559 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

22560 CHANGE HISTORY

22561 First released in Issue 3.

22562 Issue 5

22563 The DESCRIPTION is updated to indicate how an application should check for an error. This
22564 text was previously published in the APPLICATION USAGE section.

22565 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

22566 Issue 6

22567 The *lgamma()* function is no longer marked as an extension.

22568 The *lgammaf()* and *lgammal()* functions are added for alignment with the ISO/IEC 9899:1999
22569 standard.

22570 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
22571 revised to align with the ISO/IEC 9899:1999 standard.

22572 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
22573 marked.

22574 XSI extensions are marked.

22575 **NAME**

22576 link — link to a file

22577 **SYNOPSIS**

22578 #include <unistd.h>

22579 int link(const char *path1, const char *path2);

22580 **DESCRIPTION**22581 The *link()* function shall create a new link (directory entry) for the existing file, *path1*.

22582 The *path1* argument points to a pathname naming an existing file. The *path2* argument points to
 22583 a pathname naming the new directory entry to be created. The *link()* function shall atomically
 22584 create a new link for the existing file and the link count of the file shall be incremented by one.

22585 If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the
 22586 implementation supports using *link()* on directories.

22587 Upon successful completion, *link()* shall mark for update the *st_ctime* field of the file. Also, the
 22588 *st_ctime* and *st_mtime* fields of the directory that contains the new entry shall be marked for
 22589 update.

22590 If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.

22591 The implementation may require that the calling process has permission to access the existing
 22592 file.

22593 **RETURN VALUE**

22594 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 22595 indicate the error.

22596 **ERRORS**22597 The *link()* function shall fail if:

22598 [EACCES] A component of either path prefix denies search permission, or the requested
 22599 link requires writing in a directory that denies write permission, or the calling
 22600 process does not have permission to access the existing file and this is
 22601 required by the implementation.

22602 [EEXIST] The *path2* argument resolves to an existing file or refers to a symbolic link.

22603 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or
 22604 *path2* argument.

22605 [EMLINK] The number of links to the file named by *path1* would exceed {LINK_MAX}.

22606 [ENAMETOOLONG]

22607 The length of the *path1* or *path2* argument exceeds {PATH_MAX} or a
 22608 pathname component is longer than {NAME_MAX}.

22609 [ENOENT] A component of either path prefix does not exist; the file named by *path1* does
 22610 not exist; or *path1* or *path2* points to an empty string.

22611 [ENOSPC] The directory to contain the link cannot be extended.

22612 [ENOTDIR] A component of either path prefix is not a directory.

22613 [EPERM] The file named by *path1* is a directory and either the calling process does not
 22614 have appropriate privileges or the implementation prohibits using *link()* on
 22615 directories.

22616 [EROFS] The requested link requires writing in a directory on a read-only file system.

22617 [EXDEV] The link named by *path2* and the file named by *path1* are on different file

22618 systems and the implementation does not support links between file systems.

22619 XSR [EXDEV] *path1* refers to a named STREAM.

22620 The *link()* function may fail if:

22621 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during

22622 resolution of the *path1* or *path2* argument.

22623 [ENAMETOOLONG]

22624 As a result of encountering a symbolic link in resolution of the *path1* or *path2*

22625 argument, the length of the substituted pathname string exceeded

22626 {PATH_MAX}.

22627 EXAMPLES

22628 Creating a Link to a File

22629 The following example shows how to create a link to a file named **/home/cnd/mod1** by creating a

22630 new directory entry named **/modules/pass1**.

```
22631 #include <unistd.h>
22632 char *path1 = "/home/cnd/mod1";
22633 char *path2 = "/modules/pass1";
22634 int status;
22635 ...
22636 status = link (path1, path2);
```

22637 Creating a Link to a File Within a Program

22638 In the following program example, the *link()* function links the **/etc/passwd** file (defined as

22639 **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the

22640 current password file. Then, after removing the current password file (defined as

22641 **PASSWDFILE**), the new password file is saved as the current password file using the *link()*

22642 function again.

```
22643 #include <unistd.h>
22644 #define LOCKFILE "/etc/ptmp"
22645 #define PASSWDFILE "/etc/passwd"
22646 #define SAVEFILE "/etc/opasswd"
22647 ...
22648 /* Save current password file */
22649 link (PASSWDFILE, SAVEFILE);
22650
22651 /* Remove current password file. */
22652 unlink (PASSWDFILE);
22653
22654 /* Save new password file as current password file. */
22655 link (LOCKFILE, PASSWDFILE);
```

22654 APPLICATION USAGE

22655 Some implementations do allow links between file systems.

22656 RATIONALE

22657 Linking to a directory is restricted to the superuser in most historical implementations because
22658 this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This
22659 volume of IEEE Std 1003.1-2001 continues that philosophy by prohibiting *link()* and *unlink()*
22660 from doing this. Other functions could do it if the implementor designed such an extension.

22661 Some historical implementations allow linking of files on different file systems. Wording was
22662 added to explicitly allow this optional behavior.

22663 The exception for cross-file system links is intended to apply only to links that are
22664 programmatically indistinguishable from “hard” links.

22665 FUTURE DIRECTIONS

22666 None.

22667 SEE ALSO

22668 *symlink()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

22669 CHANGE HISTORY

22670 First released in Issue 1. Derived from Issue 1 of the SVID.

22671 Issue 6

22672 The following new requirements on POSIX implementations derive from alignment with the
22673 Single UNIX Specification:

- 22674 • The [ELOOP] mandatory error condition is added.
- 22675 • A second [ENAMETOOLONG] is added as an optional error condition.

22676 The following changes were made to align with the IEEE P1003.1a draft standard:

- 22677 • An explanation is added of the action when *path2* refers to a symbolic link.
- 22678 • The [ELOOP] optional error condition is added.

22679 **NAME**22680 lio_listio — list directed I/O (**REALTIME**)22681 **SYNOPSIS**

22682 AIO #include <aio.h>

```
22683       int lio_listio(int mode, struct aiocb *restrict const list[restrict],
22684           int nent, struct sigevent *restrict sig);
```

22685

22686 **DESCRIPTION**22687 The *lio_listio()* function shall initiate a list of I/O requests with a single function call.

22688 The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in <aio.h> and
 22689 determines whether the function returns when the I/O operations have been completed, or as
 22690 soon as the operations have been queued. If the *mode* argument is LIO_WAIT, the function shall
 22691 wait until all I/O is complete and the *sig* argument shall be ignored.

22692 If the *mode* argument is LIO_NOWAIT, the function shall return immediately, and asynchronous
 22693 notification shall occur, according to the *sig* argument, when all the I/O operations complete. If
 22694 *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous
 22695 notification occurs as specified in Section 2.4.1 (on page 28) when all the requests in *list* have
 22696 completed.

22697 The I/O requests enumerated by *list* are submitted in an unspecified order.

22698 The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.
 22699 The array may contain NULL elements, which shall be ignored.

22700 The *aio_lio_opcode* field of each **aiocb** structure specifies the operation to be performed. The
 22701 supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in
 22702 <aio.h>. The LIO_NOP operation causes the list entry to be ignored. If the *aio_lio_opcode*
 22703 element is equal to LIO_READ, then an I/O operation is submitted as if by a call to *aio_read()*
 22704 with the *aiocbp* equal to the address of the **aiocb** structure. If the *aio_lio_opcode* element is equal
 22705 to LIO_WRITE, then an I/O operation is submitted as if by a call to *aio_write()* with the *aiocbp*
 22706 equal to the address of the **aiocb** structure.

22707 The *aio_fildes* member specifies the file descriptor on which the operation is to be performed.22708 The *aio_buf* member specifies the address of the buffer to or from which the data is transferred.22709 The *aio_nbytes* member specifies the number of bytes of data to be transferred.

22710 The members of the **aiocb** structure further describe the I/O operation to be performed, in a
 22711 manner identical to that of the corresponding **aiocb** structure when used by the *aio_read()* and
 22712 *aio_write()* functions.

22713 The *nent* argument specifies how many elements are members of the list; that is, the length of the
 22714 array.

22715 The behavior of this function is altered according to the definitions of synchronized I/O data
 22716 integrity completion and synchronized I/O file integrity completion if synchronized I/O is
 22717 enabled on the file associated with *aio_fildes*.

22718 For regular files, no data transfer shall occur past the offset maximum established in the open
 22719 file description associated with *aiocbp->aio_fildes*.

22720 **RETURN VALUE**

22721 If the *mode* argument has the value LIO_NOWAIT, the *lio_listio()* function shall return the value
 22722 zero if the I/O operations are successfully queued; otherwise, the function shall return the value
 22723 -1 and set *errno* to indicate the error.

22724 If the *mode* argument has the value LIO_WAIT, the *lio_listio()* function shall return the value
 22725 zero when all the indicated I/O has completed successfully. Otherwise, *lio_listio()* shall return a
 22726 value of -1 and set *errno* to indicate the error.

22727 In either case, the return value only indicates the success or failure of the *lio_listio()* call itself,
 22728 not the status of the individual I/O requests. In some cases one or more of the I/O requests
 22729 contained in the list may fail. Failure of an individual request does not prevent completion of
 22730 any other individual request. To determine the outcome of each I/O request, the application
 22731 shall examine the error status associated with each **aiocb** control block. The error statuses so
 22732 returned are identical to those returned as the result of an *aio_read()* or *aio_write()* function.

22733 **ERRORS**

22734 The *lio_listio()* function shall fail if:

22735 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The
 22736 application may check the error status for each **aiocb** to determine the
 22737 individual request(s) that failed.

22738 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit
 22739 {AIO_MAX} to be exceeded.

22740 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than
 22741 {AIO_LISTIO_MAX}.

22742 [EINTR] A signal was delivered while waiting for all I/O requests to complete during
 22743 an LIO_WAIT operation. Note that, since each I/O operation invoked by
 22744 *lio_listio()* may possibly provoke a signal when it completes, this error return
 22745 may be caused by the completion of one (or more) of the very I/O operations
 22746 being awaited. Outstanding I/O requests are not canceled, and the application
 22747 shall examine each list element to determine whether the request was
 22748 initiated, canceled, or completed.

22749 [EIO] One or more of the individual I/O operations failed. The application may
 22750 check the error status for each **aiocb** structure to determine the individual
 22751 request(s) that failed.

22752 In addition to the errors returned by the *lio_listio()* function, if the *lio_listio()* function succeeds
 22753 or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list
 22754 may have been initiated. If the *lio_listio()* function fails with an error code other than [EAGAIN],
 22755 [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation
 22756 indicated by each list element can encounter errors specific to the individual read or write
 22757 function being performed. In this event, the error status for each **aiocb** control block contains the
 22758 associated error code. The error codes that can be set are the same as would be set by a *read()* or
 22759 *write()* function, with the following additional error codes possible:

22760 [EAGAIN] The requested I/O operation was not queued due to resource limitations.

22761 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 22762 *aio_cancel()* request.

22763 [EFBIG] The *aiocbp->aio_lio_opcode* is LIO_WRITE, the file is a regular file,
 22764 *aiocbp->aio_nbytes* is greater than 0, and the *aiocbp->aio_offset* is greater than or
 22765 equal to the offset maximum in the open file description associated with

22766 *aiocbp->aio_fildes*.

22767 [EINPROGRESS] The requested I/O is in progress.

22768 [EOVERFLOW] The *aiocbp->aio_lio_opcode* is LIO_READ, the file is a regular file,
 22769 *aiocbp->aio_nbytes* is greater than 0, and the *aiocbp->aio_offset* is before the
 22770 end-of-file and is greater than or equal to the offset maximum in the open file
 22771 description associated with *aiocbp->aio_fildes*.

22772 EXAMPLES

22773 None.

22774 APPLICATION USAGE

22775 None.

22776 RATIONALE

22777 Although it may appear that there are inconsistencies in the specified circumstances for error
 22778 codes, the [EIO] error condition applies when any circumstance relating to an individual
 22779 operation makes that operation fail. This might be due to a badly formulated request (for
 22780 example, the *aio_lio_opcode* field is invalid, and *aio_error()* returns [EINVAL]) or might arise from
 22781 application behavior (for example, the file descriptor is closed before the operation is initiated,
 22782 and *aio_error()* returns [EBADF]).

22783 The limitation on the set of error codes returned when operations from the list shall have been
 22784 initiated enables applications to know when operations have been started and whether
 22785 *aio_error()* is valid for a specific operation.

22786 FUTURE DIRECTIONS

22787 None.

22788 SEE ALSO

22789 *aio_read()*, *aio_write()*, *aio_error()*, *aio_return()*, *aio_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,
 22790 *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

22791 CHANGE HISTORY

22792 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

22793 Large File Summit extensions are added.

22794 Issue 6

22795 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 22796 implementation does not support the Asynchronous Input and Output option.

22797 The *lio_listio()* function is marked as part of the Asynchronous Input and Output option.

22798 The following new requirements on POSIX implementations derive from alignment with the
 22799 Single UNIX Specification:

22800 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
 22801 past the offset maximum established in the open file description associated with
 22802 *aiocbp->aio_fildes*. This change is to support large files.

22803 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support large
 22804 files.

22805 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22806 The **restrict** keyword is added to the *lio_listio()* prototype for alignment with the
 22807 ISO/IEC 9899:1999 standard.

22808 **NAME**

22809 listen — listen for socket connections and limit the queue of incoming connections

22810 **SYNOPSIS**

22811 #include <sys/socket.h>

22812 int listen(int *socket*, int *backlog*);

22813 **DESCRIPTION**

22814 The *listen()* function shall mark a connection-mode socket, specified by the *socket* argument, as
22815 accepting connections.

22816 The *backlog* argument provides a hint to the implementation which the implementation shall use
22817 to limit the number of outstanding connections in the socket's listen queue. Implementations
22818 may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*
22819 argument value shall result in a larger or equal length of the listen queue. Implementations shall
22820 support values of *backlog* up to SOMAXCONN, defined in <sys/socket.h>.

22821 The implementation may include incomplete connections in its listen queue. The limits on the
22822 number of incomplete connections and completed connections queued may be different.

22823 The implementation may have an upper limit on the length of the listen queue—either global or
22824 per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

22825 If *listen()* is called with a *backlog* argument value that is less than 0, the function behaves as if it
22826 had been called with a *backlog* argument value of 0.

22827 A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of
22828 the listen queue may be set to an implementation-defined minimum value.

22829 The socket in use may require the process to have appropriate privileges to use the *listen()*
22830 function.

22831 **RETURN VALUE**

22832 Upon successful completions, *listen()* shall return 0; otherwise, -1 shall be returned and *errno* set
22833 to indicate the error.

22834 **ERRORS**

22835 The *listen()* function shall fail if:

22836 [EBADF] The *socket* argument is not a valid file descriptor.

22837 [EDESTADDRREQ]

22838 The socket is not bound to a local address, and the protocol does not support
22839 listening on an unbound socket.

22840 [EINVAL] The *socket* is already connected.

22841 [ENOTSOCK] The *socket* argument does not refer to a socket.

22842 [EOPNOTSUPP] The socket protocol does not support *listen()*.

22843 The *listen()* function may fail if:

22844 [EACCES] The calling process does not have the appropriate privileges.

22845 [EINVAL] The *socket* has been shut down.

22846 [ENOBUFS] Insufficient resources are available in the system to complete the call.

22847 **EXAMPLES**

22848 None.

22849 **APPLICATION USAGE**

22850 None.

22851 **RATIONALE**

22852 None.

22853 **FUTURE DIRECTIONS**

22854 None.

22855 **SEE ALSO**22856 *accept()*, *connect()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>22857 **CHANGE HISTORY**

22858 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

22859 The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog*
22860 argument.

22861 **NAME**

22862 llabs — return a long integer absolute value

22863 **SYNOPSIS**

22864 #include <stdlib.h>

22865 long long llabs(long long *i*);22866 **DESCRIPTION**22867 Refer to *labs()*.

22868 NAME

22869 `lldiv` — compute quotient and remainder of a long division

22870 SYNOPSIS

22871 `#include <stdlib.h>`

22872 `lldiv_t lldiv(long long numer, long long denom);`

22873 DESCRIPTION

22874 Refer to *ldiv()*.

22875 **NAME**

22876 llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

22877 **SYNOPSIS**

22878 #include <math.h>

22879 long long llrint(double x);

22880 long long llrintf(float x);

22881 long long llrintl(long double x);

22882 **DESCRIPTION**

22883 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22884 conflict between the requirements described here and the ISO C standard is unintentional. This
 22885 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22886 These functions shall round their argument to the nearest integer value, rounding according to
 22887 the current rounding direction.

22888 An application wishing to check for error situations should set *errno* to zero and call
 22889 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 22890 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 22891 zero, an error has occurred.

22892 **RETURN VALUE**

22893 Upon successful completion, these functions shall return the rounded integer value.

22894 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.22895 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.22896 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

22897 If the correct value is positive and too large to represent as a **long long**, a domain error shall
 22898 occur and an unspecified value is returned.

22899 If the correct value is negative and too large to represent as a **long long**, a domain error shall
 22900 occur and an unspecified value is returned.

22901 **ERRORS**

22902 These functions shall fail if:

22903 MX	Domain Error	The <i>x</i> argument is NaN or $\pm$ Inf, or the correct value is not representable as an integer.

22905	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

22909 **EXAMPLES**

22910 None.

22911 **APPLICATION USAGE**

22912 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 22913 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22914 **RATIONALE**

22915 These functions provide floating-to-integer conversions. They round according to the current
 22916 rounding direction. If the rounded value is outside the range of the return type, the numeric
 22917 result is unspecified and the invalid floating-point exception is raised. When they raise no other
 22918 floating-point exception and the result differs from the argument, they raise the inexact

22919 floating-point exception.

22920 **FUTURE DIRECTIONS**

22921 None.

22922 **SEE ALSO**

22923 *feclearexcept()*, *fetestexcept()*, *lrint()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section
22924 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

22925 **CHANGE HISTORY**

22926 First released in Issue 6. Derived from the ISO / IEC 9899:1999 standard.

22927 **NAME**

22928 llround, llroundf, llroundl — round to nearest integer value

22929 **SYNOPSIS**

22930 #include <math.h>

22931 long long llround(double x);

22932 long long llroundf(float x);

22933 long long llroundl(long double x);

22934 **DESCRIPTION**

22935 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22936 conflict between the requirements described here and the ISO C standard is unintentional. This
 22937 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22938 These functions shall round their argument to the nearest integer value, rounding halfway cases
 22939 away from zero, regardless of the current rounding direction.

22940 An application wishing to check for error situations should set *errno* to zero and call
 22941 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 22942 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 22943 zero, an error has occurred.

22944 **RETURN VALUE**

22945 Upon successful completion, these functions shall return the rounded integer value.

22946 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.22947 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.22948 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

22949 If the correct value is positive and too large to represent as a **long long**, a domain error shall
 22950 occur and an unspecified value is returned.

22951 If the correct value is negative and too large to represent as a **long long**, a domain error shall
 22952 occur and an unspecified value is returned.

22953 **ERRORS**

22954 These functions shall fail if:

22955 MX	Domain Error	The <i>x</i> argument is NaN or $\pm$ Inf, or the correct value is not representable as an integer.

22957	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

22961 **EXAMPLES**

22962 None.

22963 **APPLICATION USAGE**

22964 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 22965 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22966 **RATIONALE**

22967 These functions differ from the *llrint*() functions in that the default rounding direction for the
 22968 *llround*() functions round halfway cases away from zero and need not raise the inexact floating-
 22969 point exception for non-integer arguments that round to within the range of the return type.

22970 **FUTURE DIRECTIONS**

22971 None.

22972 **SEE ALSO**

22973 *feclearexcept()*, *fetetestexcept()*, *lround()*, the Base Definitions volume of IEEE Std 1003.1-2001,
22974 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

22975 **CHANGE HISTORY**

22976 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22977 **NAME**

22978 localeconv — return locale-specific information

22979 **SYNOPSIS**

22980 #include <locale.h>

22981 struct lconv *localeconv(void);

22982 **DESCRIPTION**

22983 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 22984 conflict between the requirements described here and the ISO C standard is unintentional. This
 22985 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22986 The *localeconv()* function shall set the components of an object with the type **struct lconv** with
 22987 the values appropriate for the formatting of numeric quantities (monetary and otherwise)
 22988 according to the rules of the current locale.

22989 The members of the structure with type **char *** are pointers to strings, any of which (except
 22990 **decimal_point**) can point to " ", to indicate that the value is not available in the current locale or
 22991 is of zero length. The members with type **char** are non-negative numbers, any of which can be
 22992 {CHAR_MAX} to indicate that the value is not available in the current locale.

22993 The members include the following:

22994 **char *decimal_point**

22995 The radix character used to format non-monetary quantities.

22996 **char *thousands_sep**

22997 The character used to separate groups of digits before the decimal-point character in
 22998 formatted non-monetary quantities.

22999 **char *grouping**

23000 A string whose elements taken as one-byte integer values indicate the size of each group of
 23001 digits in formatted non-monetary quantities.

23002 **char *int_curr_symbol**

23003 The international currency symbol applicable to the current locale. The first three
 23004 characters contain the alphabetic international currency symbol in accordance with those |
 23005 specified in the ISO 4217:2001 standard. The fourth character (immediately preceding the |
 23006 null byte) is the character used to separate the international currency symbol from the
 23007 monetary quantity.

23008 **char *currency_symbol**

23009 The local currency symbol applicable to the current locale.

23010 **char *mon_decimal_point**

23011 The radix character used to format monetary quantities.

23012 **char *mon_thousands_sep**

23013 The separator for groups of digits before the decimal-point in formatted monetary
 23014 quantities.

23015 **char *mon_grouping**

23016 A string whose elements taken as one-byte integer values indicate the size of each group of
 23017 digits in formatted monetary quantities.

23018 **char *positive_sign**

23019 The string used to indicate a non-negative valued formatted monetary quantity.

23020	char *negative_sign	
23021	The string used to indicate a negative valued formatted monetary quantity.	
23022	char int_frac_digits	
23023	The number of fractional digits (those after the decimal-point) to be displayed in an	
23024	internationally formatted monetary quantity.	
23025	char frac_digits	
23026	The number of fractional digits (those after the decimal-point) to be displayed in a	
23027	formatted monetary quantity.	
23028	char p_cs_precedes	
23029	Set to 1 if the currency_symbol precedes the value for a non-negative formatted monetary	
23030	quantity. Set to 0 if the symbol succeeds the value.	
23031	char p_sep_by_space	
23032	Set to a value indicating the separation of the currency_symbol , the sign string, and the	
23033	value for a non-negative formatted monetary quantity.	
23034	char n_cs_precedes	
23035	Set to 1 if the currency_symbol precedes the value for a negative formatted monetary	
23036	quantity. Set to 0 if the symbol succeeds the value.	
23037	char n_sep_by_space	
23038	Set to a value indicating the separation of the currency_symbol , the sign string, and the	
23039	value for a negative formatted monetary quantity.	
23040	char p_sign_posn	
23041	Set to a value indicating the positioning of the positive_sign for a non-negative formatted	
23042	monetary quantity.	
23043	char n_sign_posn	
23044	Set to a value indicating the positioning of the negative_sign for a negative formatted	
23045	monetary quantity.	
23046	char int_p_cs_precedes	
23047	Set to 1 or 0 if the int_curr_symbol respectively precedes or succeeds the value for a non-	
23048	negative internationally formatted monetary quantity.	
23049	char int_n_cs_precedes	
23050	Set to 1 or 0 if the int_curr_symbol respectively precedes or succeeds the value for a	
23051	negative internationally formatted monetary quantity.	
23052	char int_p_sep_by_space	
23053	Set to a value indicating the separation of the int_curr_symbol , the sign string, and the	
23054	value for a non-negative internationally formatted monetary quantity.	
23055	char int_n_sep_by_space	
23056	Set to a value indicating the separation of the int_curr_symbol , the sign string, and the	
23057	value for a negative internationally formatted monetary quantity.	
23058	char int_p_sign_posn	
23059	Set to a value indicating the positioning of the positive_sign for a non-negative	
23060	internationally formatted monetary quantity.	
23061	char int_n_sign_posn	
23062	Set to a value indicating the positioning of the negative_sign for a negative internationally	
23063	formatted monetary quantity.	

23064	The elements of grouping and mon_grouping are interpreted according to the following:
23065	{CHAR_MAX} No further grouping is to be performed.
23066	0 The previous element is to be repeatedly used for the remainder of the digits.
23067	<i>other</i> The integer value is the number of digits that comprise the current group. The
23068	next element is examined to determine the size of the next group of digits
23069	before the current group.
23070	The values of p_sep_by_space , n_sep_by_space , int_p_sep_by_space , and int_n_sep_by_space
23071	are interpreted according to the following:
23072	0 No space separates the currency symbol and value.
23073	1 If the currency symbol and sign string are adjacent, a space separates them from the value;
23074	otherwise, a space separates the currency symbol from the value.
23075	2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a
23076	space separates the sign string from the value.
23077	For int_p_sep_by_space and int_n_sep_by_space , the fourth character of int_curr_symbol is
23078	used instead of a space.
23079	The values of p_sign_posn , n_sign_posn , int_p_sign_posn , and int_n_sign_posn are
23080	interpreted according to the following:
23081	0 Parentheses surround the quantity and currency_symbol or int_curr_symbol .
23082	1 The sign string precedes the quantity and currency_symbol or int_curr_symbol .
23083	2 The sign string succeeds the quantity and currency_symbol or int_curr_symbol .
23084	3 The sign string immediately precedes the currency_symbol or int_curr_symbol .
23085	4 The sign string immediately succeeds the currency_symbol or int_curr_symbol .
23086	The implementation shall behave as if no function in this volume of IEEE Std 1003.1-2001 calls
23087	<i>localeconv()</i> .
23088 CX	The <i>localeconv()</i> function need not be reentrant. A function that is not required to be reentrant is
23089	not required to be thread-safe.
23090	RETURN VALUE
23091	The <i>localeconv()</i> function shall return a pointer to the filled-in object. The application shall not
23092	modify the structure pointed to by the return value which may be overwritten by a subsequent
23093	call to <i>localeconv()</i> . In addition, calls to <i>setlocale()</i> with the categories <i>LC_ALL</i> , <i>LC_MONETARY</i> ,
23094	or <i>LC_NUMERIC</i> may overwrite the contents of the structure.
23095	ERRORS
23096	No errors are defined.
23097	EXAMPLES
23098	None.
23099	APPLICATION USAGE
23100	The following table illustrates the rules which may be used by four countries to format monetary
23101	quantities.

Country	Positive Format	Negative Format	International Format
Italy	L.1.230	−L.1.230	ITL.1.230
Netherlands	F 1.234,56	F −1.234,56	NLG 1.234,56
Norway	kr1.234,56	kr1.234,56−	NOK 1.234,56
Switzerland	SFr.s.1,234.56	SFr.s.1,234.56C	CHF 1,234.56

For these four countries, the respective values for the monetary members of the structure returned by `localeconv()` are:

	Italy	Netherlands	Norway	Switzerland
int_curr_symbol	"ITL. "	"NLG "	"NOK "	"CHF "
currency_symbol	"L. "	"F"	"kr"	"SFr.s. "
mon_decimal_point	" "	", "	", "	". "
mon_thousands_sep	". "	". "	". "	", "
mon_grouping	"\3 "	"\3 "	"\3 "	"\3 "
positive_sign	" "	" "	" "	" "
negative_sign	" − "	" − "	" − "	" C "
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_precedes	1	1	1	1
n_sep_by_space	0	1	0	0
p_sign_posn	1	1	1	1
n_sign_posn	1	4	2	2
int_p_cs_precedes	1	1	1	1
int_n_cs_precedes	1	1	1	1
int_p_sep_by_space	0	0	0	0
int_n_sep_by_space	0	0	0	0
int_p_sign_posn	1	1	1	1
int_n_sign_posn	1	4	4	2

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`isalpha()`, `isascii()`, `nl_langinfo()`, `printf()`, `scanf()`, `setlocale()`, `strcat()`, `strchr()`, `strcmp()`, `strcoll()`, `strcpy()`, `strftime()`, `strlen()`, `strpbrk()`, `strspn()`, `strtok()`, `strxfrm()`, `strtod()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<langinfo.h>**, **<locale.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

Issue 6

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

The RETURN VALUE section is rewritten to avoid use of the term “must”.

This reference page is updated for alignment with the ISO / IEC 9899: 1999 standard.

ISO / IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

23147 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to
23148 **int_curr_symbol** and updating the descriptions of **p_sep_by_space** and **n_sep_by_space**. These
23149 changes are for alignment with the ISO C standard.

23150 **NAME**

23151 localtime, localtime_r — convert a time value to a broken-down local time

23152 **SYNOPSIS**

23153 #include <time.h>

23154 struct tm *localtime(const time_t *timer);

23155 TSF struct tm *localtime_r(const time_t *restrict timer,

23156 struct tm *restrict result);

23157

23158 **DESCRIPTION**

23159 CX For *localtime()*: The functionality described on this reference page is aligned with the ISO C
 23160 standard. Any conflict between the requirements described here and the ISO C standard is
 23161 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23162 The *localtime()* function shall convert the time in seconds since the Epoch pointed to by *timer*
 23163 into a broken-down time, expressed as a local time. The function corrects for the timezone and
 23164 CX any seasonal time adjustments. Local timezone information is used as though *localtime()* calls
 23165 *tzset()*.

23166 The relationship between a time in seconds since the Epoch used as an argument to *localtime()*
 23167 and the **tm** structure (defined in the <time.h> header) is that the result shall be as specified in the
 23168 expression given in the definition of seconds since the Epoch (see the Base Definitions volume of
 23169 IEEE Std 1003.1-2001, Section 4.14, Seconds Since the Epoch) corrected for timezone and any
 23170 seasonal time adjustments, where the names in the structure and in the expression correspond.

23171 TSF The same relationship shall apply for *localtime_r()*.

23172 CX The *localtime()* function need not be reentrant. A function that is not required to be reentrant is
 23173 not required to be thread-safe.

23174 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 23175 objects: a broken-down time structure and an array of type **char**. Execution of any of the
 23176 functions may overwrite the information returned in either of these objects by any of the other
 23177 functions.

23178 TSF The *localtime_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*
 23179 into a broken-down time stored in the structure to which *result* points. The *localtime_r()* function
 23180 shall also return a pointer to that same structure.

23181 Unlike *localtime()*, the reentrant version is not required to set *tzname*.23182 **RETURN VALUE**

23183 Upon successful completion, the *localtime()* function shall return a pointer to the broken-down
 23184 CX time structure. If an error is detected, *localtime()* shall return a null pointer and set *errno* to
 23185 indicate the error.

23186 TSF Upon successful completion, *localtime_r()* shall return a pointer to the structure pointed to by
 23187 the argument *result*.

23188 **ERRORS**23189 The *localtime()* function shall fail if:

23190 CX [EOVERFLOW] The result cannot be represented.

23191 **EXAMPLES**23192 **Getting the Local Date and Time**

23193 The following example uses the *time()* function to calculate the time elapsed, in seconds, since
 23194 January 1, 1970 0:00 UTC (the Epoch), *localtime()* to convert that value to a broken-down time,
 23195 and *asctime()* to convert the broken-down time values into a printable string.

```
23196 #include <stdio.h>
23197 #include <time.h>

23198 int main(void)
23199 {
23200     time_t result;

23201     result = time(NULL);
23202     printf("%s%ju secs since the Epoch\n",
23203           asctime(localtime(&result)),
23204           (uintmax_t)result);
23205     return(0);
23206 }
```

23207 This example writes the current time to *stdout* in a form like this:

```
23208 Wed Jun 26 10:32:15 1996
23209 835810335 secs since the Epoch
```

23210 **Getting the Modification Time for a File**

23211 The following example gets the modification time for a file. The *localtime()* function converts the
 23212 **time_t** value of the last modification date, obtained by a previous call to *stat()*, into a **tm**
 23213 structure that contains the year, month, day, and so on.

```
23214 #include <time.h>
23215 ...
23216 struct stat statbuf;
23217 ...
23218 tm = localtime(&statbuf.st_mtime);
23219 ...
```

23220 **Timing an Event**

23221 The following example gets the current time, converts it to a string using *localtime()* and
 23222 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to
 23223 an event being timed.

```
23224 #include <time.h>
23225 #include <stdio.h>
23226 ...
23227 time_t now;
23228 int minutes_to_event;
23229 ...
23230 time(&now);
23231 printf("The time is ");
23232 fputs(asctime(localtime(&now)), stdout);
23233 printf("There are still %d minutes to the event.\n",
```

23234 minutes_to_event);
 23235 ...

23236 APPLICATION USAGE

23237 The *localtime_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 23238 possibly using a static data area that may be overwritten by each call.

23239 RATIONALE

23240 None.

23241 FUTURE DIRECTIONS

23242 None.

23243 SEE ALSO

23244 *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*,
 23245 *utime()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

23246 CHANGE HISTORY

23247 First released in Issue 1. Derived from Issue 1 of the SVID.

23248 Issue 5

23249 A note indicating that the *localtime()* function need not be reentrant is added to the
 23250 DESCRIPTION.

23251 The *localtime_r()* function is included for alignment with the POSIX Threads Extension.

23252 Issue 6

23253 The *localtime_r()* function is marked as part of the Thread-Safe Functions option.

23254 Extensions beyond the ISO C standard are marked.

23255 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 23256 its avoidance of possibly using a static data area.

23257 The **restrict** keyword is added to the *localtime_r()* prototype for alignment with the
 23258 ISO/IEC 9899:1999 standard.

23259 Examples are added.

23260 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/32 is applied, adding the
 23261 [EOverflow] error.

23262 **NAME**

23263 lockf — record locking on files

23264 **SYNOPSIS**23265 xSI `#include <unistd.h>`23266 `int lockf(int fildes, int function, off_t size);`

23267

23268 **DESCRIPTION**

23269 The *lockf()* function shall lock sections of a file with advisory-mode locks. Calls to *lockf()* from
 23270 other threads which attempt to lock the locked file section shall either return an error value or
 23271 block until the section becomes unlocked. All the locks for a process are removed when the
 23272 process terminates. Record locking with *lockf()* shall be supported for regular files and may be
 23273 supported for other files.

23274 The *fildes* argument is an open file descriptor. To establish a lock with this function, the file
 23275 descriptor shall be opened with write-only permission (O_WRONLY) or with read/write
 23276 permission (O_RDWR).

23277 The *function* argument is a control value which specifies the action to be taken. The permissible
 23278 values for *function* are defined in **<unistd.h>** as follows:

23279

23280

23281

23282

23283

23284

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

23285 F_TEST shall detect if a lock by another process is present on the specified section.

23286 F_LOCK and F_TLOCK shall both lock a section of a file if the section is available.

23287 F_ULOCK shall remove locks from a section of the file.

23288 The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be
 23289 locked or unlocked starts at the current offset in the file and extends forward for a positive size
 23290 or backward for a negative size (the preceding bytes up to but not including the current offset).
 23291 If *size* is 0, the section from the current offset through the largest possible file offset shall be
 23292 locked (that is, from the current offset through the present or any future end-of-file). An area
 23293 need not be allocated to the file to be locked because locks may exist past the end-of-file.

23294 The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained
 23295 by a previously locked section for the same process. When this occurs, or if adjacent locked
 23296 sections would occur, the sections shall be combined into a single locked section. If the request
 23297 would cause the number of locks to exceed a system-imposed limit, the request shall fail.

23298 F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available.
 23299 F_LOCK shall block the calling thread until the section is available. F_TLOCK shall cause the
 23300 function to fail if the section is already locked by another process.

23301 File locks shall be released on first close by the locking process of any file descriptor for the file.

23302 F_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the
 23303 process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or
 23304 to the end-of-file if *size* is (**off_t**)0. When all of a locked section is not released (that is, when the
 23305 beginning or end of the area to be unlocked falls within a locked section), the remaining portions
 23306 of that section shall remain locked by the process. Releasing the center portion of a locked

23307	section shall cause the remaining locked beginning and end portions to become two separate
23308	locked sections. If the request would cause the number of locks in the system to exceed a
23309	system-imposed limit, the request shall fail.
23310	A potential for deadlock occurs if the threads of a process controlling a locked section are
23311	blocked by accessing another process' locked section. If the system detects that deadlock would
23312	occur, <i>lockf()</i> shall fail with an [EDEADLK] error.
23313	The interaction between <i>fcntl()</i> and <i>lockf()</i> locks is unspecified.
23314	Blocking on a section shall be interrupted by any signal.
23315	An F_ULOCK request in which <i>size</i> is non-zero and the offset of the last byte of the requested
23316	section is the maximum value for an object of type off_t , when the process has an existing lock
23317	in which <i>size</i> is 0 and which includes the last byte of the requested section, shall be treated as a
23318	request to unlock from the start of the requested section with a size equal to 0. Otherwise, an
23319	F_ULOCK request shall attempt to unlock only the requested section.
23320	Attempting to lock a section of a file that is associated with a buffered stream produces
23321	unspecified results.
23322	RETURN VALUE
23323	Upon successful completion, <i>lockf()</i> shall return 0. Otherwise, it shall return -1, set <i>errno</i> to
23324	indicate an error, and existing locks shall not be changed.
23325	ERRORS
23326	The <i>lockf()</i> function shall fail if:
23327	[EBADF] The <i>filides</i> argument is not a valid open file descriptor; or <i>function</i> is F_LOCK
23328	or F_TLOCK and <i>filides</i> is not a valid file descriptor open for writing.
23329	[EACCES] or [EAGAIN]
23330	The <i>function</i> argument is F_TLOCK or F_TEST and the section is already
23331	locked by another process.
23332	[EDEADLK] The <i>function</i> argument is F_LOCK and a deadlock is detected.
23333	[EINTR] A signal was caught during execution of the function.
23334	[EINVAL] The <i>function</i> argument is not one of F_LOCK, F_TLOCK, F_TEST, or
23335	F_ULOCK; or <i>size</i> plus the current file offset is less than 0.
23336	[EOVERFLOW] The offset of the first, or if <i>size</i> is not 0 then the last, byte in the requested
23337	section cannot be represented correctly in an object of type off_t .
23338	The <i>lockf()</i> function may fail if:
23339	[EAGAIN] The <i>function</i> argument is F_LOCK or F_TLOCK and the file is mapped with
23340	<i>mmap()</i> .
23341	[EDEADLK] or [ENOLCK]
23342	The <i>function</i> argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request
23343	would cause the number of locks to exceed a system-imposed limit.
23344	[EOPNOTSUPP] or [EINVAL]
23345	The implementation does not support the locking of files of the type indicated
23346	by the <i>filides</i> argument.

23347 **EXAMPLES**23348 **Locking a Portion of a File**

23349 In the following example, a file named **/home/cnd/mod1** is being modified. Other processes that
 23350 use locking are prevented from changing it during this process. Only the first 10 000 bytes are
 23351 locked, and the lock call fails if another process has any part of this area locked already.

```
23352 #include <fcntl.h>
23353 #include <unistd.h>

23354 int fildes;
23355 int status;
23356 ...
23357 fildes = open("/home/cnd/mod1", O_RDWR);
23358 status = lockf(fildes, F_TLOCK, (off_t)10000);
```

23359 **APPLICATION USAGE**

23360 Record-locking should not be used in combination with the *fopen()*, *fread()*, *fwrite()*, and other
 23361 *stdio* functions. Instead, the more primitive, non-buffered functions (such as *open()*) should be
 23362 used. Unexpected results may occur in processes that do buffering in the user address space. The
 23363 process may later read/write data which is/was locked. The *stdio* functions are the most
 23364 common source of unexpected buffering.

23365 The *alarm()* function may be used to provide a timeout facility in applications requiring it.

23366 **RATIONALE**

23367 None.

23368 **FUTURE DIRECTIONS**

23369 None.

23370 **SEE ALSO**

23371 *alarm()*, *chmod()*, *close()*, *creat()*, *fcntl()*, *fopen()*, *mmap()*, *open()*, *read()*, *write()*, the Base
 23372 Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

23373 **CHANGE HISTORY**

23374 First released in Issue 4, Version 2.

23375 **Issue 5**

23376 Moved from X/OPEN UNIX extension to BASE.

23377 Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified
 23378 and moved from optional to mandatory status.

23379 A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a
 23380 file that is associated with a buffered stream.

23381 **Issue 6**

23382 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23383 **NAME**

23384 log, logf, logl — natural logarithm function

23385 **SYNOPSIS**

23386 #include <math.h>

23387 double log(double x);

23388 float logf(float x);

23389 long double logl(long double x);

23390 **DESCRIPTION**

23391 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23392 conflict between the requirements described here and the ISO C standard is unintentional. This
 23393 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23394 These functions shall compute the natural logarithm of their argument x , $\log_e(x)$.

23395 An application wishing to check for error situations should set *errno* to zero and call
 23396 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23397 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23398 zero, an error has occurred.

23399 **RETURN VALUE**23400 Upon successful completion, these functions shall return the natural logarithm of x .

23401 If x is ± 0 , a pole error shall occur and *log()*, *logf()*, and *logl()* shall return $-\text{HUGE\_VAL}$,
 23402 $-\text{HUGE\_VALF}$, and $-\text{HUGE\_VALL}$, respectively.

23403 MX For finite values of x that are less than 0, or if x is $-\text{Inf}$, a domain error shall occur, and either a
 23404 NaN (if supported), or an implementation-defined value shall be returned.

23405 MX If x is NaN, a NaN shall be returned.23406 If x is 1, +0 shall be returned.23407 If x is $+\text{Inf}$, x shall be returned.23408 **ERRORS**

23409 These functions shall fail if:

23410 MX Domain Error The finite value of x is negative, or x is $-\text{Inf}$.

23411 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23412 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 23413 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 23414 shall be raised.

23415 Pole Error The value of x is zero.

23416 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23417 then *errno* shall be set to [ERANGE]. If the integer expression
 23418 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
 23419 zero floating-point exception shall be raised.

23420 **EXAMPLES**

23421 None.

23422 **APPLICATION USAGE**

23423 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
23424 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23425 **RATIONALE**

23426 None.

23427 **FUTURE DIRECTIONS**

23428 None.

23429 **SEE ALSO**

23430 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *log10()*, *log1p()*, the Base Definitions volume of
23431 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
23432 **<math.h>**

23433 **CHANGE HISTORY**

23434 First released in Issue 1. Derived from Issue 1 of the SVID.

23435 **Issue 5**

23436 The DESCRIPTION is updated to indicate how an application should check for an error. This
23437 text was previously published in the APPLICATION USAGE section.

23438 **Issue 6**

23439 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23440 The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

23441 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
23442 revised to align with the ISO/IEC 9899:1999 standard.

23443 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
23444 marked.

23445 **NAME**

23446 log10, log10f, log10l — base 10 logarithm function

23447 **SYNOPSIS**

23448 #include <math.h>

23449 double log10(double x);

23450 float log10f(float x);

23451 long double log10l(long double x);

23452 **DESCRIPTION**

23453 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23454 conflict between the requirements described here and the ISO C standard is unintentional. This
 23455 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23456 These functions shall compute the base 10 logarithm of their argument x , $\log_{10}(x)$.

23457 An application wishing to check for error situations should set *errno* to zero and call
 23458 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23459 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23460 zero, an error has occurred.

23461 **RETURN VALUE**23462 Upon successful completion, these functions shall return the base 10 logarithm of x .

23463 If x is ± 0 , a pole error shall occur and *log10()*, *log10f()*, and *log10l()* shall return $-\text{HUGE\_VAL}$,
 23464 $-\text{HUGE\_VALF}$, and $-\text{HUGE\_VALL}$, respectively.

23465 MX For finite values of x that are less than 0, or if x is $-\text{Inf}$, a domain error shall occur, and either a
 23466 NaN (if supported), or an implementation-defined value shall be returned.

23467 MX If x is NaN, a NaN shall be returned.23468 If x is 1, +0 shall be returned.23469 If x is +Inf, +Inf shall be returned.23470 **ERRORS**

23471 These functions shall fail if:

23472 MX Domain Error The finite value of x is negative, or x is $-\text{Inf}$.

23473 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23474 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 23475 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 23476 shall be raised.

23477 Pole Error The value of x is zero.

23478 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23479 then *errno* shall be set to [ERANGE]. If the integer expression
 23480 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
 23481 zero floating-point exception shall be raised.

23482 **EXAMPLES**

23483 None.

23484 **APPLICATION USAGE**

23485 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
23486 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23487 **RATIONALE**

23488 None.

23489 **FUTURE DIRECTIONS**

23490 None.

23491 **SEE ALSO**

23492 *feclearexcept()*, *fetetestexcept()*, *isnan()*, *log()*, *pow()*, the Base Definitions volume of
23493 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
23494 **<math.h>**

23495 **CHANGE HISTORY**

23496 First released in Issue 1. Derived from Issue 1 of the SVID.

23497 **Issue 5**

23498 The DESCRIPTION is updated to indicate how an application should check for an error. This
23499 text was previously published in the APPLICATION USAGE section.

23500 **Issue 6**

23501 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23502 The *log10f()* and *log10l()* functions are added for alignment with the ISO/IEC 9899:1999
23503 standard.

23504 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
23505 revised to align with the ISO/IEC 9899:1999 standard.

23506 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
23507 marked.

23508 **NAME**

23509 log1p, log1pf, log1pl — compute a natural logarithm

23510 **SYNOPSIS**

23511 #include <math.h>

23512 double log1p(double x);

23513 float log1pf(float x);

23514 long double log1pl(long double x);

23515 **DESCRIPTION**

23516 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23517 conflict between the requirements described here and the ISO C standard is unintentional. This
 23518 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23519 These functions shall compute $\log_e(1.0 + x)$.

23520 An application wishing to check for error situations should set *errno* to zero and call
 23521 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23522 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23523 zero, an error has occurred.

23524 **RETURN VALUE**23525 Upon successful completion, these functions shall return the natural logarithm of $1.0 + x$.

23526 If x is -1 , a pole error shall occur and *log1p()*, *log1pf()*, and *log1pl()* shall return $-\text{HUGE\_VAL}$,
 23527 $-\text{HUGE\_VALF}$, and $-\text{HUGE\_VALL}$, respectively.

23528 MX For finite values of x that are less than -1 , or if x is $-\text{Inf}$, a domain error shall occur, and either a
 23529 NaN (if supported), or an implementation-defined value shall be returned.

23530 MX If x is NaN, a NaN shall be returned.23531 If x is ± 0 , or $+\text{Inf}$, x shall be returned.23532 If x is subnormal, a range error may occur and x should be returned.23533 **ERRORS**

23534 These functions shall fail if:

23535 MX Domain Error The finite value of x is less than -1 , or x is $-\text{Inf}$.

23536 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23537 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 23538 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 23539 shall be raised.

23540 Pole Error The value of x is -1 .

23541 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23542 then *errno* shall be set to [ERANGE]. If the integer expression
 23543 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
 23544 zero floating-point exception shall be raised.

23545 These functions may fail if:

23546 MX Range Error The value of x is subnormal.

23547 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23548 then *errno* shall be set to [ERANGE]. If the integer expression
 23549 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 23550 floating-point exception shall be raised.

23551 **EXAMPLES**

23552 None.

23553 **APPLICATION USAGE**

23554 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
23555 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23556 **RATIONALE**

23557 None.

23558 **FUTURE DIRECTIONS**

23559 None.

23560 **SEE ALSO**

23561 *feclearexcept()*, *fetestexcept()*, *log()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section
23562 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

23563 **CHANGE HISTORY**

23564 First released in Issue 4, Version 2.

23565 **Issue 5**

23566 Moved from X/OPEN UNIX extension to BASE.

23567 **Issue 6**

23568 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23569 The *log1p()* function is no longer marked as an extension.

23570 The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999
23571 standard.

23572 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
23573 revised to align with the ISO/IEC 9899:1999 standard.

23574 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
23575 marked.

23576 **NAME**

23577 log2, log2f, log2l — compute base 2 logarithm functions

23578 **SYNOPSIS**

23579 #include <math.h>

23580 double log2(double x);

23581 float log2f(float x);

23582 long double log2l(long double x);

23583 **DESCRIPTION**

23584 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23585 conflict between the requirements described here and the ISO C standard is unintentional. This
 23586 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23587 These functions shall compute the base 2 logarithm of their argument x , $\log_2(x)$.

23588 An application wishing to check for error situations should set *errno* to zero and call
 23589 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23590 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23591 zero, an error has occurred.

23592 **RETURN VALUE**23593 Upon successful completion, these functions shall return the base 2 logarithm of x .

23594 If x is ± 0 , a pole error shall occur and *log2()*, *log2f()*, and *log2l()* shall return $-\text{HUGE\_VAL}$,
 23595 $-\text{HUGE\_VALF}$, and $-\text{HUGE\_VALL}$, respectively.

23596 MX For finite values of x that are less than 0, or if x is $-\text{Inf}$, a domain error shall occur, and either a
 23597 NaN (if supported), or an implementation-defined value shall be returned.

23598 MX If x is NaN, a NaN shall be returned.23599 If x is 1, +0 shall be returned.23600 If x is $+\text{Inf}$, x shall be returned.23601 **ERRORS**

23602 These functions shall fail if:

23603 MX Domain Error The finite value of x is less than zero, or x is $-\text{Inf}$.

23604 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23605 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 23606 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 23607 shall be raised.

23608 Pole Error The value of x is zero.

23609 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23610 then *errno* shall be set to [ERANGE]. If the integer expression
 23611 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
 23612 zero floating-point exception shall be raised.

23613 **EXAMPLES**

23614 None.

23615 **APPLICATION USAGE**

23616 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
23617 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23618 **RATIONALE**

23619 None.

23620 **FUTURE DIRECTIONS**

23621 None.

23622 **SEE ALSO**

23623 *feclearexcept()*, *fetetestexcept()*, *log()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section
23624 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

23625 **CHANGE HISTORY**

23626 First released in Issue 6. Derived from the ISO / IEC 9899:1999 standard.

23627 **NAME**

23628 logb, logbf, logbl — radix-independent exponent

23629 **SYNOPSIS**

23630 #include <math.h>

23631 double logb(double x);

23632 float logbf(float x);

23633 long double logbl(long double x);

23634 **DESCRIPTION**

23635 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 23636 conflict between the requirements described here and the ISO C standard is unintentional. This
 23637 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23638 These functions shall compute the exponent of x , which is the integral part of $\log_r |x|$, as a
 23639 signed floating-point value, for non-zero x , where r is the radix of the machine's floating-point
 23640 arithmetic, which is the value of FLT_RADIX defined in the <float.h> header.

23641 If x is subnormal it is treated as though it were normalized; thus for finite positive x :

23642 $1 \leq x * \text{FLT\_RADIX}^{-\text{logb}(x)} < \text{FLT\_RADIX}$

23643 An application wishing to check for error situations should set *errno* to zero and call
 23644 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23645 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23646 zero, an error has occurred.

23647 **RETURN VALUE**

23648 Upon successful completion, these functions shall return the exponent of x .

23649 If x is ± 0 , a pole error shall occur and *logb*(), *logbf*(), and *logbl*() shall return *-HUGE_VAL*,
 23650 *-HUGE_VALF*, and *-HUGE_VALL*, respectively.

23651 **MX** If x is NaN, a NaN shall be returned.

23652 If x is $\pm\text{Inf}$, *+Inf* shall be returned.

23653 **ERRORS**

23654 These functions shall fail if:

23655 Pole Error The value of x is ± 0 .

23656 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 23657 then *errno* shall be set to [ERANGE]. If the integer expression
 23658 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
 23659 zero floating-point exception shall be raised.

23660 **EXAMPLES**

23661 None.

23662 **APPLICATION USAGE**

23663 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 23664 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23665 **RATIONALE**

23666 None.

23667 **FUTURE DIRECTIONS**

23668 None.

23669 **SEE ALSO**

23670 *feclearexcept()*, *fetestexcept()*, *ilogb()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-2001,
23671 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<float.h>**, **<math.h>**

23672 **CHANGE HISTORY**

23673 First released in Issue 4, Version 2.

23674 **Issue 5**

23675 Moved from X/OPEN UNIX extension to BASE.

23676 **Issue 6**23677 The *logb()* function is no longer marked as an extension.23678 The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

23679 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
23680 revised to align with the ISO/IEC 9899:1999 standard.

23681 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
23682 marked.

23683 **NAME**

23684 logf, logl — natural logarithm function

23685 **SYNOPSIS**

23686 #include <math.h>

23687 float logf(float x);

23688 long double logl(long double x);

23689 **DESCRIPTION**23690 Refer to *log()*.

23691 **NAME**

23692 longjmp — non-local goto

23693 **SYNOPSIS**

23694 #include <setjmp.h>

23695 void longjmp(jmp_buf env, int val);

23696 **DESCRIPTION**

23697 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23698 conflict between the requirements described here and the ISO C standard is unintentional. This
 23699 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23700 The *longjmp()* function shall restore the environment saved by the most recent invocation of
 23701 *setjmp()* in the same thread, with the corresponding **jmp_buf** argument. If there is no such
 23702 invocation, or if the function containing the invocation of *setjmp()* has terminated execution in
 23703 the interim, or if the invocation of *setjmp()* was within the scope of an identifier with variably
 23704 CX modified type and execution has left that scope in the interim, the behavior is undefined. It is
 23705 unspecified whether *longjmp()* restores the signal mask, leaves the signal mask unchanged, or
 23706 restores it to its value at the time *setjmp()* was called.

23707 All accessible objects have values, and all other components of the abstract machine have state
 23708 (for example, floating-point status flags and open files), as of the time *longjmp()* was called,
 23709 except that the values of objects of automatic storage duration are unspecified if they meet all
 23710 the following conditions:

- 23711 • They are local to the function containing the corresponding *setjmp()* invocation.
- 23712 • They do not have volatile-qualified type.
- 23713 • They are changed between the *setjmp()* invocation and *longjmp()* call.

23714 CX As it bypasses the usual function call and return mechanisms, *longjmp()* shall execute correctly
 23715 in contexts of interrupts, signals, and any of their associated functions. However, if *longjmp()* is
 23716 invoked from a nested signal handler (that is, from a function invoked as a result of a signal
 23717 raised during the handling of another signal), the behavior is undefined.

23718 The effect of a call to *longjmp()* where initialization of the **jmp_buf** structure was not performed
 23719 in the calling thread is undefined.

23720 **RETURN VALUE**

23721 After *longjmp()* is completed, program execution continues as if the corresponding invocation of
 23722 *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause
 23723 *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

23724 **ERRORS**

23725 No errors are defined.

23726 **EXAMPLES**

23727 None.

23728 **APPLICATION USAGE**

23729 Applications whose behavior depends on the value of the signal mask should not use *longjmp()*
 23730 and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the
 23731 *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under
 23732 application control).

23733 RATIONALE

23734 None.

23735 FUTURE DIRECTIONS

23736 None.

23737 SEE ALSO

23738 *setjmp()*, *sigaction()*, *siglongjmp()*, *sigsetjmp()*, the Base Definitions volume of
23739 IEEE Std 1003.1-2001, <**setjmp.h**>

23740 CHANGE HISTORY

23741 First released in Issue 1. Derived from Issue 1 of the SVID.

23742 Issue 5

23743 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

23744 Issue 6

23745 Extensions beyond the ISO C standard are marked.

23746 The following new requirements on POSIX implementations derive from alignment with the
23747 Single UNIX Specification:

- 23748 • The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask unspecified.

23749 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

23750 **NAME**

23751 lrand48 — generate uniformly distributed pseudo-random non-negative long integers

23752 **SYNOPSIS**

23753 xSI #include <stdlib.h>

23754 long lrand48(void);

23755

23756 **DESCRIPTION**

23757 Refer to *drand48()*.

23758 **NAME**

23759 lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

23760 **SYNOPSIS**

23761 #include <math.h>

23762 long lrint(double x);

23763 long lrintf(float x);

23764 long lrintl(long double x);

23765 **DESCRIPTION**

23766 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23767 conflict between the requirements described here and the ISO C standard is unintentional. This
 23768 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23769 These functions shall round their argument to the nearest integer value, rounding according to
 23770 the current rounding direction.

23771 An application wishing to check for error situations should set *errno* to zero and call
 23772 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23773 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23774 zero, an error has occurred.

23775 **RETURN VALUE**

23776 Upon successful completion, these functions shall return the rounded integer value.

23777 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.23778 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.23779 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

23780 If the correct value is positive and too large to represent as a **long**, a domain error shall occur
 23781 and an unspecified value is returned.

23782 If the correct value is negative and too large to represent as a **long**, a domain error shall occur
 23783 and an unspecified value is returned.

23784 **ERRORS**

23785 These functions shall fail if:

23786 MX	Domain Error	The <i>x</i> argument is NaN or ±Inf, or the correct value is not representable as an integer.

23788	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, 23789 then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling 23790 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception 23791 shall be raised.

23792 **EXAMPLES**

23793 None.

23794 **APPLICATION USAGE**

23795 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 23796 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23797 **RATIONALE**

23798 These functions provide floating-to-integer conversions. They round according to the current
 23799 rounding direction. If the rounded value is outside the range of the return type, the numeric
 23800 result is unspecified and the invalid floating-point exception is raised. When they raise no other
 23801 floating-point exception and the result differs from the argument, they raise the inexact

23802 floating-point exception.

23803 **FUTURE DIRECTIONS**

23804 None.

23805 **SEE ALSO**

23806 *feclearexcept()*, *fetestexcept()*, *lrint()*, the Base Definitions volume of IEEE Std 1003.1-2001,
23807 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

23808 **CHANGE HISTORY**

23809 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23810 **NAME**

23811 lround, lroundf, lroundl — round to nearest integer value

23812 **SYNOPSIS**

23813 #include <math.h>

23814 long lround(double x);

23815 long lroundf(float x);

23816 long lroundl(long double x);

23817 **DESCRIPTION**

23818 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23819 conflict between the requirements described here and the ISO C standard is unintentional. This
 23820 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23821 These functions shall round their argument to the nearest integer value, rounding halfway cases
 23822 away from zero, regardless of the current rounding direction.

23823 An application wishing to check for error situations should set *errno* to zero and call
 23824 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23825 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23826 zero, an error has occurred.

23827 **RETURN VALUE**

23828 Upon successful completion, these functions shall return the rounded integer value.

23829 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.23830 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.23831 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

23832 If the correct value is positive and too large to represent as a **long**, a domain error shall occur
 23833 and an unspecified value is returned.

23834 If the correct value is negative and too large to represent as a **long**, a domain error shall occur
 23835 and an unspecified value is returned.

23836 **ERRORS**

23837 These functions shall fail if:

23838 MX	Domain Error	The <i>x</i> argument is NaN or $\pm$ Inf, or the correct value is not representable as an integer.

23840	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

23844 **EXAMPLES**

23845 None.

23846 **APPLICATION USAGE**

23847 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 23848 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23849 **RATIONALE**

23850 These functions differ from the *lrint()* functions in the default rounding direction, with the
 23851 *lround()* functions rounding halfway cases away from zero and needing not to raise the inexact
 23852 floating-point exception for non-integer arguments that round to within the range of the return
 23853 type.

23854 **FUTURE DIRECTIONS**

23855 None.

23856 **SEE ALSO**

23857 *feclearexcept()*, *fetetestexcept()*, *llround()*, the Base Definitions volume of IEEE Std 1003.1-2001,
23858 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

23859 **CHANGE HISTORY**

23860 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23861 **NAME**

23862 lsearch, lfind — linear search and update

23863 **SYNOPSIS**

23864 xSI #include <search.h>

```

23865 void *lsearch(const void *key, void *base, size_t *nel, size_t width,
23866             int (*compar)(const void *, const void *));
23867 void *lfind(const void *key, const void *base, size_t *nel,
23868            size_t width, int (*compar)(const void *, const void *));
23869 
```

23870 **DESCRIPTION**

23871 The *lsearch()* function shall linearly search the table and return a pointer into the table for the
 23872 matching entry. If the entry does not occur, it shall be added at the end of the table. The *key*
 23873 argument points to the entry to be sought in the table. The *base* argument points to the first
 23874 element in the table. The *width* argument is the size of an element in bytes. The *nel* argument
 23875 points to an integer containing the current number of elements in the table. The integer to which
 23876 *nel* points shall be incremented if the entry is added to the table. The *compar* argument points to
 23877 a comparison function which the application shall supply (for example, *strcmp()*). It is called
 23878 with two arguments that point to the elements being compared. The application shall ensure
 23879 that the function returns 0 if the elements are equal, and non-zero otherwise.

23880 The *lfind()* function shall be equivalent to *lsearch()*, except that if the entry is not found, it is not
 23881 added to the table. Instead, a null pointer is returned.

23882 **RETURN VALUE**

23883 If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it. Otherwise,
 23884 *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly added element.

23885 Both functions shall return a null pointer in case of error.

23886 **ERRORS**

23887 No errors are defined.

23888 **EXAMPLES**23889 **Storing Strings in a Table**

23890 This fragment reads in less than or equal to TABSIZE strings of length less than or equal to
 23891 ELSIZE and stores them in a table, eliminating duplicates.

```

23892 #include <stdio.h>
23893 #include <string.h>
23894 #include <search.h>
23895 #define TABSIZE 50
23896 #define ELSIZE 120
23897 ...
23898     char line[ELSIZE], tab[TABSIZE][ELSIZE];
23899     size_t nel = 0;
23900     ...
23901     while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
23902         (void) lsearch(line, tab, &nel,
23903                       ELSIZE, (int (*)(const void *, const void *)) strcmp);
23904     ...

```

23905 Finding a Matching Entry

23906 The following example finds any line that reads "This is a test.".

```
23907 #include <search.h>
23908 #include <string.h>
23909 ...
23910 char line[ELSIZE], tab[TABSIZE][ELSIZE];
23911 size_t nel = 0;
23912 char *findline;
23913 void *entry;

23914 findline = "This is a test.\n";

23915 entry = lfind(findline, tab, &nel, ELSIZE, (
23916     int (*)(const void *, const void *)) strcmp);
```

23917 APPLICATION USAGE

23918 The comparison function need not compare every byte, so arbitrary data may be contained in
23919 the elements in addition to the values being compared.

23920 Undefined results can occur if there is not enough room in the table to add a new item.

23921 RATIONALE

23922 None.

23923 FUTURE DIRECTIONS

23924 None.

23925 SEE ALSO

23926 *hcreate()*, *tsearch()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<search.h>**

23927 CHANGE HISTORY

23928 First released in Issue 1. Derived from Issue 1 of the SVID.

23929 Issue 6

23930 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23931 **NAME**

23932 lseek — move the read/write file offset

23933 **SYNOPSIS**

23934 #include <unistd.h>

23935 off_t lseek(int *fildev*, off_t *offset*, int *whence*);23936 **DESCRIPTION**23937 The *lseek()* function shall set the file offset for the open file description associated with the file
23938 descriptor *fildev*, as follows:

- 23939 • If *whence* is SEEK_SET, the file offset shall be set to *offset* bytes.
- 23940 • If *whence* is SEEK_CUR, the file offset shall be set to its current location plus *offset*.
- 23941 • If *whence* is SEEK_END, the file offset shall be set to the size of the file plus *offset*.

23942 The symbolic constants SEEK_SET, SEEK_CUR, and SEEK_END are defined in <unistd.h>.

23943 The behavior of *lseek()* on devices which are incapable of seeking is implementation-defined.
23944 The value of the file offset associated with such a device is undefined.23945 The *lseek()* function shall allow the file offset to be set beyond the end of the existing data in the
23946 file. If data is later written at this point, subsequent reads of data in the gap shall return bytes
23947 with the value 0 until data is actually written into the gap.23948 The *lseek()* function shall not, by itself, extend the size of a file.23949 SHM If *fildev* refers to a shared memory object, the result of the *lseek()* function is unspecified.23950 TYM If *fildev* refers to a typed memory object, the result of the *lseek()* function is unspecified.23951 **RETURN VALUE**23952 Upon successful completion, the resulting offset, as measured in bytes from the beginning of the
23953 file, shall be returned. Otherwise, (**off_t**)-1 shall be returned, *errno* shall be set to indicate the
23954 error, and the file offset shall remain unchanged.23955 **ERRORS**23956 The *lseek()* function shall fail if:

23957	[EBADF]	The <i>fildev</i> argument is not an open file descriptor.
23958	[EINVAL]	The <i>whence</i> argument is not a proper value, or the resulting file offset would
23959		be negative for a regular file, block special file, or directory.
23960	[EOVERFLOW]	The resulting file offset would be a value which cannot be represented
23961		correctly in an object of type off_t .
23962	[ESPIPE]	The <i>fildev</i> argument is associated with a pipe, FIFO, or socket.

23963 **EXAMPLES**

23964 None.

23965 **APPLICATION USAGE**

23966 None.

23967 **RATIONALE**23968 The ISO C standard includes the functions *fgetpos()* and *fsetpos()*, which work on very large files
23969 by use of a special positioning type.23970 Although *lseek()* may position the file offset beyond the end of the file, this function does not
23971 itself extend the size of the file. While the only function in IEEE Std 1003.1-2001 that may directly

23972 extend the size of the file is *write()*, *truncate()*, and *ftruncate()*, several functions originally
 23973 derived from the ISO C standard, such as *fwrite()*, *fprintf()*, and so on, may do so (by causing
 23974 calls on *write()*).

23975 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-
 23976 defined and device-dependent (for example, memory may have few invalid values). A negative
 23977 file offset may be valid for some devices in some implementations.

23978 The POSIX.1-1990 standard did not specifically prohibit *lseek()* from returning a negative offset.
 23979 Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return
 23980 to determine whether a return value of (**off_t**)−1 is a negative offset or an indication of an error
 23981 condition. The standard developers did not wish to require this action on the part of a
 23982 conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting
 23983 file offset would be negative for a regular file, block special file, or directory.

23984 **FUTURE DIRECTIONS**

23985 None.

23986 **SEE ALSO**

23987 *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

23988 **CHANGE HISTORY**

23989 First released in Issue 1. Derived from Issue 1 of the SVID.

23990 **Issue 5**

23991 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

23992 Large File Summit extensions are added.

23993 **Issue 6**

23994 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

23995 The following new requirements on POSIX implementations derive from alignment with the
 23996 Single UNIX Specification:

23997 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
 23998 required for conforming implementations of previous POSIX specifications, it was not
 23999 required for UNIX applications.

24000 • The [EOVERFLOW] error condition is added. This change is to support large files.

24001 An additional [ESPIPE] error condition is added for sockets.

24002 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 24003 *lseek()* results are unspecified for typed memory objects.

24004 **NAME**

24005 lstat — get symbolic link status

24006 **SYNOPSIS**

24007 #include <sys/stat.h>

24008 int lstat(const char *restrict *path*, struct stat *restrict *buf*);24009 **DESCRIPTION**

24010 The *lstat()* function shall be equivalent to *stat()*, except when *path* refers to a symbolic link. In
 24011 that case *lstat()* shall return information about the link, while *stat()* shall return information
 24012 about the file the link references.

24013 For symbolic links, the *st_mode* member shall contain meaningful information when used with
 24014 the file type macros, and the *st_size* member shall contain the length of the pathname contained
 24015 in the symbolic link. File mode bits and the contents of the remaining members of the **stat**
 24016 structure are unspecified. The value returned in the *st_size* member is the length of the contents
 24017 of the symbolic link, and does not count any trailing null.

24018 **RETURN VALUE**

24019 Upon successful completion, *lstat()* shall return 0. Otherwise, it shall return -1 and set *errno* to
 24020 indicate the error.

24021 **ERRORS**24022 The *lstat()* function shall fail if:

24023 [EACCES] A component of the path prefix denies search permission.

24024 [EIO] An error occurred while reading from the file system.

24025 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 24026 argument.

24027 [ENAMETOOLONG]

24028 The length of a pathname exceeds {PATH_MAX} or a pathname component is
 24029 longer than {NAME_MAX}.

24030 [ENOTDIR] A component of the path prefix is not a directory.

24031 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

24032 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
 24033 serial number cannot be represented correctly in the structure pointed to by
 24034 *buf*.

24035 The *lstat()* function may fail if:

24036 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 24037 resolution of the *path* argument.

24038 [ENAMETOOLONG]

24039 As a result of encountering a symbolic link in resolution of the *path* argument,
 24040 the length of the substituted pathname string exceeded {PATH_MAX}.

24041 [EOVERFLOW] One of the members is too large to store into the structure pointed to by the
 24042 *buf* argument.

24043 **EXAMPLES**24044 **Obtaining Symbolic Link Status Information**

24045 The following example shows how to obtain status information for a symbolic link named
24046 **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path*
24047 argument specified the filename for the file pointed to by the symbolic link (**/home/cnd/mod1**),
24048 the results of calling the function would be the same as those returned by a call to the *stat()*
24049 function.

```
24050 #include <sys/stat.h>
24051 struct stat buffer;
24052 int status;
24053 ...
24054 status = lstat("/modules/pass1", &buffer);
```

24055 **APPLICATION USAGE**

24056 None.

24057 **RATIONALE**

24058 The *lstat()* function is not required to update the time-related fields if the named file is not a
24059 symbolic link. While the *st_uid*, *st_gid*, *st_atime*, *st_mtime*, and *st_ctime* members of the **stat**
24060 structure may apply to a symbolic link, they are not required to do so. No functions in
24061 IEEE Std 1003.1-2001 are required to maintain any of these time fields.

24062 **FUTURE DIRECTIONS**

24063 None.

24064 **SEE ALSO**

24065 *fstat()*, *readlink()*, *stat()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-2001,
24066 **<sys/stat.h>**

24067 **CHANGE HISTORY**

24068 First released in Issue 4, Version 2.

24069 **Issue 5**

24070 Moved from X/OPEN UNIX extension to BASE.

24071 Large File Summit extensions are added.

24072 **Issue 6**

24073 The following changes were made to align with the IEEE P1003.1a draft standard:

- 24074 • This function is now mandatory.
- 24075 • The [ELOOP] optional error condition is added.

24076 The **restrict** keyword is added to the *lstat()* prototype for alignment with the ISO/IEC 9899:1999
24077 standard.

24078 **NAME**

24079 makecontext, swapcontext — manipulate user contexts

24080 **SYNOPSIS**

```

24081 xSI      #include <ucontext.h>

24082          void makecontext(ucontext_t *ucp, void (*func)(void),
24083                          int argc, ...);
24084          int swapcontext(ucontext_t *restrict oucp,
24085                          const ucontext_t *restrict ucp);
24086

```

24087 **DESCRIPTION**

24088 The *makecontext()* function shall modify the context specified by *ucp*, which has been initialized
 24089 using *getcontext()*. When this context is resumed using *swapcontext()* or *setcontext()*, program
 24090 execution shall continue by calling *func*, passing it the arguments that follow *argc* in the
 24091 *makecontext()* call.

24092 Before a call is made to *makecontext()*, the application shall ensure that the context being
 24093 modified has a stack allocated for it. The application shall ensure that the value of *argc* matches
 24094 the number of arguments of type **int** passed to *func*; otherwise, the behavior is undefined.

24095 The *uc_link* member is used to determine the context that shall be resumed when the context
 24096 being modified by *makecontext()* returns. The application shall ensure that the *uc_link* member is
 24097 initialized prior to the call to *makecontext()*.

24098 The *swapcontext()* function shall save the current context in the context structure pointed to by
 24099 *oucp* and shall set the context to the context structure pointed to by *ucp*.

24100 **RETURN VALUE**

24101 Upon successful completion, *swapcontext()* shall return 0. Otherwise, -1 shall be returned and
 24102 *errno* set to indicate the error.

24103 **ERRORS**

24104 The *swapcontext()* function shall fail if:

24105 [ENOMEM] The *ucp* argument does not have enough stack left to complete the operation.

24106 **EXAMPLES**

24107 The following example illustrates the use of *makecontext()*:

```

24108          #include <stdio.h>
24109          #include <ucontext.h>

24110          static ucontext_t ctx[3];

24111          static void
24112          f1 (void)
24113          {
24114              puts("start f1");
24115              swapcontext(&ctx[1], &ctx[2]);
24116              puts("finish f1");
24117          }

24118          static void
24119          f2 (void)
24120          {
24121              puts("start f2");
24122              swapcontext(&ctx[2], &ctx[1]);

```

```

24123         puts("finish f2");
24124     }
24125     int
24126     main (void)
24127     {
24128         char st1[8192];
24129         char st2[8192];
24130
24131         getcontext(&ctx[1]);
24132         ctx[1].uc_stack.ss_sp = st1;
24133         ctx[1].uc_stack.ss_size = sizeof st1;
24134         ctx[1].uc_link = &ctx[0];
24135         makecontext(&ctx[1], f1, 0);
24136
24137         getcontext(&ctx[2]);
24138         ctx[2].uc_stack.ss_sp = st2;
24139         ctx[2].uc_stack.ss_size = sizeof st2;
24140         ctx[2].uc_link = &ctx[1];
24141         makecontext(&ctx[2], f2, 0);
24142
24143         swapcontext(&ctx[0], &ctx[2]);
24144         return 0;
24145     }

```

24143 APPLICATION USAGE

24144 None.

24145 RATIONALE

24146 None.

24147 FUTURE DIRECTIONS

24148 None.

24149 SEE ALSO

24150 *exit()*, *getcontext()*, *sigaction()*, *sigprocmask()*, the Base Definitions volume of
 24151 IEEE Std 1003.1-2001, <**ucontext.h**>

24152 CHANGE HISTORY

24153 First released in Issue 4, Version 2.

24154 Issue 5

24155 Moved from X/OPEN UNIX extension to BASE.

24156 In the ERRORS section, the description of [ENOMEM] is changed to apply to *swapcontext()* only.

24157 Issue 6

24158 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

24159 The **restrict** keyword is added to the *swapcontext()* prototype for alignment with the
 24160 ISO/IEC 9899:1999 standard.

24161 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/33 is applied, clarifying that the
 24162 arguments passed to *func* are of type **int**.

24163 NAME

24164 malloc — a memory allocator

24165 SYNOPSIS

24166 #include <stdlib.h>

24167 void *malloc(size_t size);

24168 DESCRIPTION

24169 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24170 conflict between the requirements described here and the ISO C standard is unintentional. This
24171 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24172 The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by
24173 *size* and whose value is unspecified.

24174 The order and contiguity of storage allocated by successive calls to *malloc()* is unspecified. The
24175 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
24176 a pointer to any type of object and then used to access such an object in the space allocated (until
24177 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
24178 disjoint from any other object. The pointer returned points to the start (lowest byte address) of
24179 the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of
24180 the space requested is 0, the behavior is implementation-defined: the value returned shall be
24181 either a null pointer or a unique pointer.

24182 RETURN VALUE

24183 Upon successful completion with *size* not equal to 0, *malloc()* shall return a pointer to the
24184 allocated space. If *size* is 0, either a null pointer or a unique pointer that can be successfully
24185 CX passed to *free()* shall be returned. Otherwise, it shall return a null pointer and set *errno* to
24186 indicate the error.

24187 ERRORS

24188 The *malloc()* function shall fail if:

24189 CX [ENOMEM] Insufficient storage space is available.

24190 EXAMPLES

24191 None.

24192 APPLICATION USAGE

24193 None.

24194 RATIONALE

24195 None.

24196 FUTURE DIRECTIONS

24197 None.

24198 SEE ALSO

24199 *calloc()*, *free()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

24200 CHANGE HISTORY

24201 First released in Issue 1. Derived from Issue 1 of the SVID.

24202 Issue 6

24203 Extensions beyond the ISO C standard are marked.

24204 The following new requirements on POSIX implementations derive from alignment with the
24205 Single UNIX Specification:

- 24206 • In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
- 24207 • The [ENOMEM] error condition is added.

24208 **NAME**

24209 mblen — get number of bytes in a character

24210 **SYNOPSIS**

24211 #include <stdlib.h>

24212 int mblen(const char *s, size_t n);

24213 **DESCRIPTION**

24214 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24215 conflict between the requirements described here and the ISO C standard is unintentional. This
 24216 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24217 If s is not a null pointer, *mblen()* shall determine the number of bytes constituting the character
 24218 pointed to by s. Except that the shift state of *mbtowc()* is not affected, it shall be equivalent to:

24219 mbtowc((wchar_t *)0, s, n);

24220 The implementation shall behave as if no function defined in this volume of
 24221 IEEE Std 1003.1-2001 calls *mblen()*.

24222 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 24223 state-dependent encoding, this function shall be placed into its initial state by a call for which its
 24224 character pointer argument, s, is a null pointer. Subsequent calls with s as other than a null
 24225 pointer shall cause the internal state of the function to be altered as necessary. A call with s as a
 24226 null pointer shall cause this function to return a non-zero value if encodings have state
 24227 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift
 24228 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an
 24229 adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be
 24230 unspecified.

24231 **RETURN VALUE**

24232 If s is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,
 24233 respectively, do or do not have state-dependent encodings. If s is not a null pointer, *mblen()* shall
 24234 either return 0 (if s points to the null byte), or return the number of bytes that constitute the
 24235 character (if the next n or fewer bytes form a valid character), or return -1 (if they do not form a
 24236 CX valid character) and may set *errno* to indicate the error. In no case shall the value returned be
 24237 greater than n or the value of the {MB_CUR_MAX} macro.

24238 **ERRORS**24239 The *mblen()* function may fail if:

24240 XSI [EILSEQ] Invalid character sequence is detected.

24241 **EXAMPLES**

24242 None.

24243 **APPLICATION USAGE**

24244 None.

24245 **RATIONALE**

24246 None.

24247 **FUTURE DIRECTIONS**

24248 None.

24249 **SEE ALSO**

24250 *mbtowc()*, *mbstowcs()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of
24251 IEEE Std 1003.1-2001, **<stdlib.h>**

24252 **CHANGE HISTORY**

24253 First released in Issue 4. Aligned with the ISO C standard.

24254 NAME

24255 mbrlen — get number of bytes in a character (restartable)

24256 SYNOPSIS

24257 #include <wchar.h>

24258 size_t mbrlen(const char *restrict s, size_t n,
24259 mbstate_t *restrict ps);

24260 DESCRIPTION

24261 cx The functionality described on this reference page is aligned with the ISO C standard. Any
24262 conflict between the requirements described here and the ISO C standard is unintentional. This
24263 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24264 If *s* is not a null pointer, *mbrlen()* shall determine the number of bytes constituting the character
24265 pointed to by *s*. It shall be equivalent to:

24266 mbstate_t internal;
24267 mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);

24268 If *ps* is a null pointer, the *mbrlen()* function shall use its own internal **mbstate_t** object, which is
24269 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
24270 pointed to by *ps* shall be used to completely describe the current conversion state of the
24271 associated character sequence. The implementation shall behave as if no function defined in this
24272 volume of IEEE Std 1003.1-2001 calls *mbrlen()*.

24273 The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

24274 RETURN VALUE

24275 The *mbrlen()* function shall return the first of the following that applies:

24276	0	If the next <i>n</i> or fewer bytes complete the character that corresponds to the null wide character.
24277		
24278	<i>positive</i>	If the next <i>n</i> or fewer bytes complete a valid character; the value returned shall be the number of bytes that complete the character.
24279		
24280	(size_t)−2	If the next <i>n</i> bytes contribute to an incomplete but potentially valid character, and all <i>n</i> bytes have been processed. When <i>n</i> has at least the value of the {MB_CUR_MAX} macro, this case can only occur if <i>s</i> points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).
24281		
24282		
24283		
24284		
24285	(size_t)−1	If an encoding error occurs, in which case the next <i>n</i> or fewer bytes do not contribute to a complete and valid character. In this case, [EILSEQ] shall be stored in <i>errno</i> and the conversion state is undefined.
24286		
24287		

24288 ERRORS

24289 The *mbrlen()* function may fail if:

24290	[EINVAL]	<i>ps</i> points to an object that contains an invalid conversion state.
24291	[EILSEQ]	Invalid character sequence is detected.

24292 **EXAMPLES**

24293 None.

24294 **APPLICATION USAGE**

24295 None.

24296 **RATIONALE**

24297 None.

24298 **FUTURE DIRECTIONS**

24299 None.

24300 **SEE ALSO**24301 *mbsinit()*, *mbrtowc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>24302 **CHANGE HISTORY**24303 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
24304 (E).24305 **Issue 6**24306 The *mbrlen()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24307 **NAME**

24308 mbrtowc — convert a character to a wide-character code (restartable)

24309 **SYNOPSIS**

24310 #include <wchar.h>

```
24311       size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,
24312                      size_t n, mbstate_t *restrict ps);
```

24313 **DESCRIPTION**

24314 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 24315 conflict between the requirements described here and the ISO C standard is unintentional. This
 24316 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24317 If *s* is a null pointer, the *mbrtowc()* function shall be equivalent to the call:

```
24318     mbrtowc(NULL, "", 1, ps)
```

24319 In this case, the values of the arguments *pwc* and *n* are ignored.

24320 If *s* is not a null pointer, the *mbrtowc()* function shall inspect at most *n* bytes beginning at the
 24321 byte pointed to by *s* to determine the number of bytes needed to complete the next character
 24322 (including any shift sequences). If the function determines that the next character is completed, it
 24323 shall determine the value of the corresponding wide character and then, if *pwc* is not a null
 24324 pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide
 24325 character is the null wide character, the resulting state described shall be the initial conversion
 24326 state.

24327 If *ps* is a null pointer, the *mbrtowc()* function shall use its own internal **mbstate_t** object, which
 24328 shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t**
 24329 object pointed to by *ps* shall be used to completely describe the current conversion state of the
 24330 associated character sequence. The implementation shall behave as if no function defined in this
 24331 volume of IEEE Std 1003.1-2001 calls *mbrtowc()*.

24332 The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

24333 **RETURN VALUE**

24334 The *mbrtowc()* function shall return the first of the following that applies:

24335 0 If the next *n* or fewer bytes complete the character that corresponds to the null
 24336 wide character (which is the value stored).

24337 between 1 and *n* inclusive

24338 If the next *n* or fewer bytes complete a valid character (which is the value
 24339 stored); the value returned shall be the number of bytes that complete the
 24340 character.

24341 (**size_t**)−2 If the next *n* bytes contribute to an incomplete but potentially valid character,
 24342 and all *n* bytes have been processed (no value is stored). When *n* has at least
 24343 the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at
 24344 a sequence of redundant shift sequences (for implementations with state-
 24345 dependent encodings).

24346 (**size_t**)−1 If an encoding error occurs, in which case the next *n* or fewer bytes do not
 24347 contribute to a complete and valid character (no value is stored). In this case,
 24348 [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

24349 **ERRORS**

24350 The *mbrtowc()* function may fail if:

24351 **CX** [EINVAL] *ps* points to an object that contains an invalid conversion state.

24352 [EILSEQ] Invalid character sequence is detected.

24353 **EXAMPLES**

24354 None.

24355 **APPLICATION USAGE**

24356 None.

24357 **RATIONALE**

24358 None.

24359 **FUTURE DIRECTIONS**

24360 None.

24361 **SEE ALSO**

24362 *mbstowc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>

24363 **CHANGE HISTORY**

24364 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
24365 (E).

24366 **Issue 6**

24367 The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24368 The following new requirements on POSIX implementations derive from alignment with the
24369 Single UNIX Specification:

- 24370 • The [EINVAL] error condition is added.

24371 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

24372 **NAME**

24373 mbsinit — determine conversion object status

24374 **SYNOPSIS**

24375 #include <wchar.h>

24376 int mbsinit(const mbstate_t *ps);

24377 **DESCRIPTION**

24378 cx The functionality described on this reference page is aligned with the ISO C standard. Any
24379 conflict between the requirements described here and the ISO C standard is unintentional. This
24380 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24381 If *ps* is not a null pointer, the *mbsinit()* function shall determine whether the object pointed to by
24382 *ps* describes an initial conversion state.

24383 **RETURN VALUE**

24384 The *mbsinit()* function shall return non-zero if *ps* is a null pointer, or if the pointed-to object
24385 describes an initial conversion state; otherwise, it shall return zero.

24386 If an **mbstate_t** object is altered by any of the functions described as “restartable”, and is then
24387 used with a different character sequence, or in the other conversion direction, or with a different
24388 *LC_CTYPE* category setting than on earlier function calls, the behavior is undefined.

24389 **ERRORS**

24390 No errors are defined.

24391 **EXAMPLES**

24392 None.

24393 **APPLICATION USAGE**

24394 The **mbstate_t** object is used to describe the current conversion state from a particular character
24395 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the
24396 *LC_CTYPE* category of the current locale.

24397 The initial conversion state corresponds, for a conversion in either direction, to the beginning of
24398 a new character sequence in the initial shift state. A zero valued **mbstate_t** object is at least one
24399 way to describe an initial conversion state. A zero valued **mbstate_t** object can be used to initiate
24400 conversion involving any character sequence, in any *LC_CTYPE* category setting.

24401 **RATIONALE**

24402 None.

24403 **FUTURE DIRECTIONS**

24404 None.

24405 **SEE ALSO**

24406 *mbrlen()*, *mbrtowc()*, *wcrtomb()*, *mbsrtowcs()*, *wcsrtombs()*, the Base Definitions volume of
24407 IEEE Std 1003.1-2001, <**wchar.h**>

24408 **CHANGE HISTORY**

24409 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
24410 (E).

24411 **NAME**

24412 mbsrtowcs — convert a character string to a wide-character string (restartable)

24413 **SYNOPSIS**

24414 #include <wchar.h>

```
24415       size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
24416                       size_t len, mbstate_t *restrict ps);
```

24417 **DESCRIPTION**

24418 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24419 conflict between the requirements described here and the ISO C standard is unintentional. This
 24420 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24421 The *mbsrtowcs()* function shall convert a sequence of characters, beginning in the conversion
 24422 state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a
 24423 sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters
 24424 shall be stored into the array pointed to by *dst*. Conversion continues up to and including a
 24425 terminating null character, which shall also be stored. Conversion shall stop early in either of the
 24426 following cases:

- 24427 • A sequence of bytes is encountered that does not form a valid character.
- 24428 • *len* codes have been stored into the array pointed to by *dst* (and *dst* is not a null pointer).

24429 Each conversion shall take place as if by a call to the *mbrtowc()* function.

24430 If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null
 24431 pointer (if conversion stopped due to reaching a terminating null character) or the address just
 24432 past the last character converted (if any). If conversion stopped due to reaching a terminating
 24433 null character, and if *dst* is not a null pointer, the resulting state described shall be the initial
 24434 conversion state.

24435 If *ps* is a null pointer, the *mbsrtowcs()* function shall use its own internal **mbstate_t** object, which
 24436 is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
 24437 pointed to by *ps* shall be used to completely describe the current conversion state of the
 24438 associated character sequence. The implementation behaves as if no function defined in this
 24439 volume of IEEE Std 1003.1-2001 calls *mbsrtowcs()*.

24440 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

24441 **RETURN VALUE**

24442 If the input conversion encounters a sequence of bytes that do not form a valid character, an
 24443 encoding error occurs. In this case, the *mbsrtowcs()* function stores the value of the macro
 24444 [EILSEQ] in *errno* and shall return (**size_t**)-1; the conversion state is undefined. Otherwise, it
 24445 shall return the number of characters successfully converted, not including the terminating null
 24446 (if any).

24447 **ERRORS**

24448 The *mbsrtowcs()* function may fail if:

24449 CX	[EINVAL] <i>ps</i> points to an object that contains an invalid conversion state.
24450	[EILSEQ] Invalid character sequence is detected.

24451 **EXAMPLES**

24452 None.

24453 **APPLICATION USAGE**

24454 None.

24455 **RATIONALE**

24456 None.

24457 **FUTURE DIRECTIONS**

24458 None.

24459 **SEE ALSO**24460 *mbsinit()*, *mbrtowc()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**24461 **CHANGE HISTORY**24462 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
24463 (E).24464 **Issue 6**24465 The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24466 The [EINVAL] error condition is marked CX.

24467 **NAME**

24468 mbstowcs — convert a character string to a wide-character string

24469 **SYNOPSIS**

24470 #include <stdlib.h>

24471 size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,
24472 size_t n);

24473 **DESCRIPTION**

24474 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24475 conflict between the requirements described here and the ISO C standard is unintentional. This
24476 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24477 The *mbstowcs()* function shall convert a sequence of characters that begins in the initial shift
24478 state from the array pointed to by *s* into a sequence of corresponding wide-character codes and
24479 shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No
24480 characters that follow a null byte (which is converted into a wide-character code with value 0)
24481 shall be examined or converted. Each character shall be converted as if by a call to *mbtowc()*,
24482 except that the shift state of *mbtowc()* is not affected.

24483 No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes
24484 place between objects that overlap, the behavior is undefined.

24485 XSI The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale. If
24486 *pwcs* is a null pointer, *mbstowcs()* shall return the length required to convert the entire array
24487 regardless of the value of *n*, but no values are stored.

24488 **RETURN VALUE**

24489 CX If an invalid character is encountered, *mbstowcs()* shall return **(size_t)-1** and may set *errno* to
24490 indicate the error.

24491 XSI Otherwise, *mbstowcs()* shall return the number of the array elements modified (or required if
24492 *pwcs* is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if
24493 the value returned is *n*.

24494 **ERRORS**24495 The *mbstowcs()* function may fail if:

24496 XSI [EILSEQ] Invalid byte sequence is detected.

24497 **EXAMPLES**

24498 None.

24499 **APPLICATION USAGE**

24500 None.

24501 **RATIONALE**

24502 None.

24503 **FUTURE DIRECTIONS**

24504 None.

24505 **SEE ALSO**

24506 *mblen()*, *mbtowc()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-2001,
24507 <stdlib.h>

24508 **CHANGE HISTORY**

24509 First released in Issue 4. Aligned with the ISO C standard.

24510 **Issue 6**

24511 The *mbstowcs()* prototype is updated for alignment with the ISO / IEC 9899:1999 standard.

24512 Extensions beyond the ISO C standard are marked.

24513 **NAME**

24514 mbtowc — convert a character to a wide-character code

24515 **SYNOPSIS**

24516 #include <stdlib.h>

24517 int mbtowc(wchar_t *restrict *pwc*, const char *restrict *s*, size_t *n*);24518 **DESCRIPTION**

24519 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24520 conflict between the requirements described here and the ISO C standard is unintentional. This
 24521 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24522 If *s* is not a null pointer, *mbtowc()* shall determine the number of bytes that constitute the
 24523 character pointed to by *s*. It shall then determine the wide-character code for the value of type
 24524 **wchar_t** that corresponds to that character. (The value of the wide-character code corresponding
 24525 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* shall store
 24526 the wide-character code in the object pointed to by *pwc*.

24527 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 24528 state-dependent encoding, this function is placed into its initial state by a call for which its
 24529 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 24530 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 24531 null pointer shall cause this function to return a non-zero value if encodings have state
 24532 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift
 24533 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an
 24534 adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be
 24535 unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

24536 The implementation shall behave as if no function defined in this volume of
 24537 IEEE Std 1003.1-2001 calls *mbtowc()*.

24538 **RETURN VALUE**

24539 If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings,
 24540 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()*
 24541 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the
 24542 CX converted character (if the next *n* or fewer bytes form a valid character), or return -1 and may
 24543 set *errno* to indicate the error (if they do not form a valid character).

24544 In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

24545 **ERRORS**

24546 The *mbtowc()* function may fail if:

24547 XSI [EILSEQ] Invalid character sequence is detected.

24548 **EXAMPLES**

24549 None.

24550 **APPLICATION USAGE**

24551 None.

24552 **RATIONALE**

24553 None.

24554 **FUTURE DIRECTIONS**

24555 None.

24556 **SEE ALSO**

24557 *mblen()*, *mbstowcs()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-2001,
24558 **<stdlib.h>**

24559 **CHANGE HISTORY**

24560 First released in Issue 4. Aligned with the ISO C standard.

24561 **Issue 6**

24562 The *mbtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24563 Extensions beyond the ISO C standard are marked.

24564 **NAME**

24565 memccpy — copy bytes in memory

24566 **SYNOPSIS**

24567 xSI #include <string.h>

24568 void *memccpy(void *restrict s1, const void *restrict s2,
24569 int c, size_t n);

24570

24571 **DESCRIPTION**

24572 The *memccpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first
24573 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,
24574 whichever comes first. If copying takes place between objects that overlap, the behavior is
24575 undefined.

24576 **RETURN VALUE**

24577 The *memccpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null
24578 pointer if *c* was not found in the first *n* bytes of *s2*.

24579 **ERRORS**

24580 No errors are defined.

24581 **EXAMPLES**

24582 None.

24583 **APPLICATION USAGE**24584 The *memccpy()* function does not check for the overflow of the receiving memory area.24585 **RATIONALE**

24586 None.

24587 **FUTURE DIRECTIONS**

24588 None.

24589 **SEE ALSO**24590 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24591 **CHANGE HISTORY**

24592 First released in Issue 1. Derived from Issue 1 of the SVID.

24593 **Issue 6**

24594 The **restrict** keyword is added to the *memccpy()* prototype for alignment with the
24595 ISO/IEC 9899:1999 standard.

24596 **NAME**

24597 memchr — find byte in memory

24598 **SYNOPSIS**

24599 #include <string.h>

24600 void *memchr(const void *s, int c, size_t n);

24601 **DESCRIPTION**

24602 cx The functionality described on this reference page is aligned with the ISO C standard. Any
24603 conflict between the requirements described here and the ISO C standard is unintentional. This
24604 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24605 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in
24606 the initial *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s*.

24607 **RETURN VALUE**

24608 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte does
24609 not occur in the object.

24610 **ERRORS**

24611 No errors are defined.

24612 **EXAMPLES**

24613 None.

24614 **APPLICATION USAGE**

24615 None.

24616 **RATIONALE**

24617 None.

24618 **FUTURE DIRECTIONS**

24619 None.

24620 **SEE ALSO**24621 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24622 **CHANGE HISTORY**

24623 First released in Issue 1. Derived from Issue 1 of the SVID.

24624 **NAME**

24625 memcmp — compare bytes in memory

24626 **SYNOPSIS**

24627 #include <string.h>

24628 int memcmp(const void *s1, const void *s2, size_t n);

24629 **DESCRIPTION**

24630 cx The functionality described on this reference page is aligned with the ISO C standard. Any
24631 conflict between the requirements described here and the ISO C standard is unintentional. This
24632 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24633 The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the
24634 object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

24635 The sign of a non-zero return value shall be determined by the sign of the difference between the
24636 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects
24637 being compared.

24638 **RETURN VALUE**

24639 The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object
24640 pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

24641 **ERRORS**

24642 No errors are defined.

24643 **EXAMPLES**

24644 None.

24645 **APPLICATION USAGE**

24646 None.

24647 **RATIONALE**

24648 None.

24649 **FUTURE DIRECTIONS**

24650 None.

24651 **SEE ALSO**24652 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24653 **CHANGE HISTORY**

24654 First released in Issue 1. Derived from Issue 1 of the SVID.

24655 **NAME**

24656 memcpy — copy bytes in memory

24657 **SYNOPSIS**

24658 #include <string.h>

24659 void *memcpy(void *restrict *s1*, const void *restrict *s2*, size_t *n*);24660 **DESCRIPTION**

24661 cx The functionality described on this reference page is aligned with the ISO C standard. Any
24662 conflict between the requirements described here and the ISO C standard is unintentional. This
24663 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24664 The *memcpy()* function shall copy *n* bytes from the object pointed to by *s2* into the object pointed
24665 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

24666 **RETURN VALUE**24667 The *memcpy()* function shall return *s1*; no return value is reserved to indicate an error.24668 **ERRORS**

24669 No errors are defined.

24670 **EXAMPLES**

24671 None.

24672 **APPLICATION USAGE**24673 The *memcpy()* function does not check for the overflow of the receiving memory area.24674 **RATIONALE**

24675 None.

24676 **FUTURE DIRECTIONS**

24677 None.

24678 **SEE ALSO**24679 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24680 **CHANGE HISTORY**

24681 First released in Issue 1. Derived from Issue 1 of the SVID.

24682 **Issue 6**24683 The *memcpy()* prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

24684 **NAME**

24685 memmove — copy bytes in memory with overlapping areas

24686 **SYNOPSIS**

24687 #include <string.h>

24688 void *memmove(void *s1, const void *s2, size_t n);

24689 **DESCRIPTION**

24690 cx The functionality described on this reference page is aligned with the ISO C standard. Any
24691 conflict between the requirements described here and the ISO C standard is unintentional. This
24692 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24693 The *memmove()* function shall copy *n* bytes from the object pointed to by *s2* into the object
24694 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first
24695 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and
24696 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

24697 **RETURN VALUE**24698 The *memmove()* function shall return *s1*; no return value is reserved to indicate an error.24699 **ERRORS**

24700 No errors are defined.

24701 **EXAMPLES**

24702 None.

24703 **APPLICATION USAGE**

24704 None.

24705 **RATIONALE**

24706 None.

24707 **FUTURE DIRECTIONS**

24708 None.

24709 **SEE ALSO**24710 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24711 **CHANGE HISTORY**

24712 First released in Issue 4. Derived from the ANSI C standard.

24713 NAME

24714 memset — set bytes in memory

24715 SYNOPSIS

24716 #include <string.h>

24717 void *memset(void *s, int c, size_t n);

24718 DESCRIPTION

24719 cx The functionality described on this reference page is aligned with the ISO C standard. Any
24720 conflict between the requirements described here and the ISO C standard is unintentional. This
24721 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24722 The *memset()* function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes
24723 of the object pointed to by *s*.

24724 RETURN VALUE

24725 The *memset()* function shall return *s*; no return value is reserved to indicate an error.

24726 ERRORS

24727 No errors are defined.

24728 EXAMPLES

24729 None.

24730 APPLICATION USAGE

24731 None.

24732 RATIONALE

24733 None.

24734 FUTURE DIRECTIONS

24735 None.

24736 SEE ALSO

24737 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>

24738 CHANGE HISTORY

24739 First released in Issue 1. Derived from Issue 1 of the SVID.

24740 **NAME**

24741 mkdir — make a directory

24742 **SYNOPSIS**

24743 #include <sys/stat.h>

24744 int mkdir(const char *path, mode_t mode);

24745 **DESCRIPTION**

24746 The *mkdir()* function shall create a new directory with name *path*. The file permission bits of the
 24747 new directory shall be initialized from *mode*. These file permission bits of the *mode* argument
 24748 shall be modified by the process' file creation mask.

24749 When bits in *mode* other than the file permission bits are set, the meaning of these additional bits
 24750 is implementation-defined.

24751 The directory's user ID shall be set to the process' effective user ID. The directory's group ID
 24752 shall be set to the group ID of the parent directory or to the effective group ID of the process.
 24753 Implementations shall provide a way to initialize the directory's group ID to the group ID of the
 24754 parent directory. Implementations may, but need not, provide an implementation-defined way
 24755 to initialize the directory's group ID to the effective group ID of the calling process.

24756 The newly created directory shall be an empty directory.

24757 If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

24758 Upon successful completion, *mkdir()* shall mark for update the *st_atime*, *st_ctime*, and *st_mtime*
 24759 fields of the directory. Also, the *st_ctime* and *st_mtime* fields of the directory that contains the
 24760 new entry shall be marked for update.

24761 **RETURN VALUE**

24762 Upon successful completion, *mkdir()* shall return 0. Otherwise, -1 shall be returned, no directory
 24763 shall be created, and *errno* shall be set to indicate the error.

24764 **ERRORS**

24765 The *mkdir()* function shall fail if:

24766 [EACCES] Search permission is denied on a component of the path prefix, or write
 24767 permission is denied on the parent directory of the directory to be created.

24768 [EEXIST] The named file exists.

24769 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 24770 argument.

24771 [EMLINK] The link count of the parent directory would exceed {LINK_MAX}.

24772 [ENAMETOOLONG]

24773 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 24774 component is longer than {NAME_MAX}.

24775 [ENOENT] A component of the path prefix specified by *path* does not name an existing
 24776 directory or *path* is an empty string.

24777 [ENOSPC] The file system does not contain enough space to hold the contents of the new
 24778 directory or to extend the parent directory of the new directory.

24779 [ENOTDIR] A component of the path prefix is not a directory.

24780 [EROFS] The parent directory resides on a read-only file system.

24781 The *mkdir()* function may fail if:

24782 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
24783 resolution of the *path* argument.

24784 [ENAMETOOLONG]

24785 As a result of encountering a symbolic link in resolution of the *path* argument,
24786 the length of the substituted pathname string exceeded {PATH_MAX}.

24787 EXAMPLES

24788 Creating a Directory

24789 The following example shows how to create a directory named **/home/cnd/mod1**, with
24790 read/write/search permissions for owner and group, and with read/search permissions for
24791 others.

```
24792 #include <sys/types.h>
```

```
24793 #include <sys/stat.h>
```

```
24794 int status;
```

```
24795 ...
```

```
24796 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

24797 APPLICATION USAGE

24798 None.

24799 RATIONALE

24800 The *mkdir()* function originated in 4.2 BSD and was added to System V in Release 3.0.

24801 4.3 BSD detects [ENAMETOOLONG].

24802 The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the
24803 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2
24804 required that implementations provide a way to have the group ID be set to the group ID of the
24805 containing directory, but did not prohibit implementations also supporting a way to set the
24806 group ID to the effective group ID of the creating process. Conforming applications should not
24807 assume which group ID will be used. If it matters, an application can use *chown()* to set the
24808 group ID after the directory is created, or determine under what conditions the implementation
24809 will set the desired group ID.

24810 FUTURE DIRECTIONS

24811 None.

24812 SEE ALSO

24813 *umask()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**, **<sys/types.h>**

24814 CHANGE HISTORY

24815 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

24816 Issue 6

24817 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

24818 The following new requirements on POSIX implementations derive from alignment with the
24819 Single UNIX Specification:

- 24820 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
24821 required for conforming implementations of previous POSIX specifications, it was not
24822 required for UNIX applications.

24823 • The [ELOOP] mandatory error condition is added.

24824 • A second [ENAMETOOLONG] is added as an optional error condition.

24825 The following changes were made to align with the IEEE P1003.1a draft standard:

24826 • The [ELOOP] optional error condition is added.

24827 **NAME**

24828 mkfifo — make a FIFO special file

24829 **SYNOPSIS**

24830 #include <sys/stat.h>

24831 int mkfifo(const char *path, mode_t mode);

24832 **DESCRIPTION**

24833 The *mkfifo()* function shall create a new FIFO special file named by the pathname pointed to by
 24834 *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission
 24835 bits of the *mode* argument shall be modified by the process' file creation mask.

24836 When bits in *mode* other than the file permission bits are set, the effect is implementation-
 24837 defined.

24838 If *path* names a symbolic link, *mkfifo()* shall fail and set *errno* to [EEXIST].

24839 The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set
 24840 to the group ID of the parent directory or to the effective group ID of the process.
 24841 Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the
 24842 parent directory. Implementations may, but need not, provide an implementation-defined way
 24843 to initialize the FIFO's group ID to the effective group ID of the calling process.

24844 Upon successful completion, *mkfifo()* shall mark for update the *st_atime*, *st_ctime*, and *st_mtime*
 24845 fields of the file. Also, the *st_ctime* and *st_mtime* fields of the directory that contains the new
 24846 entry shall be marked for update.

24847 **RETURN VALUE**

24848 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, no FIFO shall
 24849 be created, and *errno* shall be set to indicate the error.

24850 **ERRORS**24851 The *mkfifo()* function shall fail if:

24852 [EACCES] A component of the path prefix denies search permission, or write permission
 24853 is denied on the parent directory of the FIFO to be created.

24854 [EEXIST] The named file already exists.

24855 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 24856 argument.

24857 [ENAMETOOLONG]

24858 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 24859 component is longer than {NAME_MAX}.

24860 [ENOENT] A component of the path prefix specified by *path* does not name an existing
 24861 directory or *path* is an empty string.

24862 [ENOSPC] The directory that would contain the new file cannot be extended or the file
 24863 system is out of file-allocation resources.

24864 [ENOTDIR] A component of the path prefix is not a directory.

24865 [EROFS] The named file resides on a read-only file system.

24866 The *mkfifo()* function may fail if:

24867 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 24868 resolution of the *path* argument.

24869 [ENAMETOOLONG]

24870 As a result of encountering a symbolic link in resolution of the *path* argument,
 24871 the length of the substituted pathname string exceeded {PATH_MAX}.

24872 EXAMPLES

24873 Creating a FIFO File

24874 The following example shows how to create a FIFO file named `/home/cnd/mod_done`, with
 24875 read/write permissions for owner, and with read permissions for group and others.

```
24876 #include <sys/types.h>
24877 #include <sys/stat.h>

24878 int status;
24879 ...
24880 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
24881               S_IRGRP | S_IROTH);
```

24882 APPLICATION USAGE

24883 None.

24884 RATIONALE

24885 The syntax of this function is intended to maintain compatibility with historical
 24886 implementations of *mknod()*. The latter function was included in the 1984 /usr/group standard
 24887 but only for use in creating FIFO special files. The *mknod()* function was originally excluded
 24888 from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir()* and
 24889 *mkfifo()*. The *mknod()* function is now included for alignment with the Single UNIX
 24890 Specification.

24891 The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the
 24892 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2
 24893 required that implementations provide a way to have the group ID be set to the group ID of the
 24894 containing directory, but did not prohibit implementations also supporting a way to set the
 24895 group ID to the effective group ID of the creating process. Conforming applications should not
 24896 assume which group ID will be used. If it matters, an application can use *chown()* to set the
 24897 group ID after the FIFO is created, or determine under what conditions the implementation will
 24898 set the desired group ID.

24899 FUTURE DIRECTIONS

24900 None.

24901 SEE ALSO

24902 *umask()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<sys/stat.h>`, `<sys/types.h>`

24903 CHANGE HISTORY

24904 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

24905 Issue 6

24906 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

24907 The following new requirements on POSIX implementations derive from alignment with the
 24908 Single UNIX Specification:

- 24909 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
 24910 required for conforming implementations of previous POSIX specifications, it was not
 24911 required for UNIX applications.

24912 • The [ELOOP] mandatory error condition is added.

24913 • A second [ENAMETOOLONG] is added as an optional error condition.

24914 The following changes were made to align with the IEEE P1003.1a draft standard:

24915 • The [ELOOP] optional error condition is added.

24916 **NAME**

24917 mknod — make a directory, a special file, or a regular file

24918 **SYNOPSIS**24919 xSI `#include <sys/stat.h>`24920 `int mknod(const char *path, mode_t mode, dev_t dev);`

24921

24922 **DESCRIPTION**24923 The *mknod()* function shall create a new file named by the pathname to which the argument *path*
24924 points.24925 The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the
24926 following symbolic constants:

24927

24928

24929

24930

24931

24932

24933

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

24934 The only portable use of *mknod()* is to create a FIFO-special file. If *mode* is not S_IFIFO or *dev* is
24935 not 0, the behavior of *mknod()* is unspecified.24936 The permissions for the new file are OR'ed into the *mode* argument, and may be selected from
24937 any combination of the following symbolic constants:

24938

24939

24940

24941

24942

24943

24944

24945

24946

24947

24948

24949

24950

24951

24952

24953

24954

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

24955 The user ID of the file shall be initialized to the effective user ID of the process. The group ID of
 24956 the file shall be initialized to either the effective group ID of the process or the group ID of the
 24957 parent directory. Implementations shall provide a way to initialize the file's group ID to the
 24958 group ID of the parent directory. Implementations may, but need not, provide an
 24959 implementation-defined way to initialize the file's group ID to the effective group ID of the
 24960 calling process. The owner, group, and other permission bits of *mode* shall be modified by the file
 24961 mode creation mask of the process. The *mknod()* function shall clear each bit whose
 24962 corresponding bit in the file mode creation mask of the process is set.

24963	If <i>path</i> names a symbolic link, <i>mknod()</i> shall fail and set <i>errno</i> to [EEXIST].	
24964	Upon successful completion, <i>mknod()</i> shall mark for update the <i>st_atime</i> , <i>st_ctime</i> , and <i>st_mtime</i>	
24965	fields of the file. Also, the <i>st_ctime</i> and <i>st_mtime</i> fields of the directory that contains the new	
24966	entry shall be marked for update.	
24967	Only a process with appropriate privileges may invoke <i>mknod()</i> for file types other than FIFO-	
24968	special.	
24969	RETURN VALUE	
24970	Upon successful completion, <i>mknod()</i> shall return 0. Otherwise, it shall return -1 , the new file	
24971	shall not be created, and <i>errno</i> shall be set to indicate the error.	
24972	ERRORS	
24973	The <i>mknod()</i> function shall fail if:	
24974	[EACCES]	A component of the path prefix denies search permission, or write permission
24975		is denied on the parent directory.
24976	[EEXIST]	The named file exists.
24977	[EINVAL]	An invalid argument exists.
24978	[EIO]	An I/O error occurred while accessing the file system.
24979	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i>
24980		argument.
24981	[ENAMETOOLONG]	
24982		The length of a pathname exceeds {PATH_MAX} or a pathname component is
24983		longer than {NAME_MAX}.
24984	[ENOENT]	A component of the path prefix specified by <i>path</i> does not name an existing
24985		directory or <i>path</i> is an empty string.
24986	[ENOSPC]	The directory that would contain the new file cannot be extended or the file
24987		system is out of file allocation resources.
24988	[ENOTDIR]	A component of the path prefix is not a directory.
24989	[EPERM]	The invoking process does not have appropriate privileges and the file type is
24990		not FIFO-special.
24991	[EROFS]	The directory in which the file is to be created is located on a read-only file
24992		system.
24993	The <i>mknod()</i> function may fail if:	
24994	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
24995		resolution of the <i>path</i> argument.
24996	[ENAMETOOLONG]	
24997		Pathname resolution of a symbolic link produced an intermediate result
24998		whose length exceeds {PATH_MAX}.

24999 **EXAMPLES**25000 **Creating a FIFO Special File**

25001 The following example shows how to create a FIFO special file named `/home/cnd/mod_done`,
 25002 with read/write permissions for owner, and with read permissions for group and others.

```
25003 #include <sys/types.h>
25004 #include <sys/stat.h>

25005 dev_t dev;
25006 int status;
25007 ...
25008 status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
25009               S_IRUSR | S_IRGRP | S_IROTH, dev);
```

25010 **APPLICATION USAGE**

25011 The `mkfifo()` function is preferred over this function for making FIFO special files.

25012 **RATIONALE**

25013 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group
 25014 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required
 25015 that implementations provide a way to have the group ID be set to the group ID of the
 25016 containing directory, but did not prohibit implementations also supporting a way to set the
 25017 group ID to the effective group ID of the creating process. Conforming applications should not
 25018 assume which group ID will be used. If it matters, an application can use `chown()` to set the
 25019 group ID after the file is created, or determine under what conditions the implementation will
 25020 set the desired group ID.

25021 **FUTURE DIRECTIONS**

25022 None.

25023 **SEE ALSO**

25024 `chmod()`, `creat()`, `exec`, `mkdir()`, `mkfifo()`, `open()`, `stat()`, `umask()`, the Base Definitions volume of
 25025 IEEE Std 1003.1-2001, **<sys/stat.h>**

25026 **CHANGE HISTORY**

25027 First released in Issue 4, Version 2.

25028 **Issue 5**

25029 Moved from X/OPEN UNIX extension to BASE.

25030 **Issue 6**

25031 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25032 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 25033 [ELOOP] error condition is added.

25034 **NAME**

25035 mkstemp — make a unique filename

25036 **SYNOPSIS**25037 xSI `#include <stdlib.h>`25038 `int mkstemp(char *template);`

25039

25040 **DESCRIPTION**

25041 The *mkstemp()* function shall replace the contents of the string pointed to by *template* by a unique
25042 filename, and return a file descriptor for the file open for reading and writing. The function thus
25043 prevents any possible race condition between testing whether the file exists and opening it for
25044 use. The string in *template* should look like a filename with six trailing 'X's; *mkstemp()* replaces
25045 each 'X' with a character from the portable filename character set. The characters are chosen
25046 such that the resulting name does not duplicate the name of an existing file at the time of a call
25047 to *mkstemp()*.

25048 **RETURN VALUE**

25049 Upon successful completion, *mkstemp()* shall return an open file descriptor. Otherwise, `-1` shall
25050 be returned if no suitable file could be created.

25051 **ERRORS**

25052 No errors are defined.

25053 **EXAMPLES**25054 **Generating a Filename**

25055 The following example creates a file with a 10-character name beginning with the characters
25056 "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file
25057 descriptor that identifies the file.

```
25058 #include <stdlib.h>
25059 ...
25060 char template[] = "/tmp/fileXXXXXX";
25061 int fd;
25062 fd = mkstemp(template);
```

25063 **APPLICATION USAGE**

25064 It is possible to run out of letters.

25065 The *mkstemp()* function need not check to determine whether the filename part of *template*
25066 exceeds the maximum allowable filename length.

25067 **RATIONALE**

25068 None.

25069 **FUTURE DIRECTIONS**

25070 None.

25071 **SEE ALSO**

25072 *getpid()*, *open()*, *tmpfile()*, *tmpnam()*, the Base Definitions volume of IEEE Std 1003.1-2001,
25073 `<stdlib.h>`

25074 **CHANGE HISTORY**

25075 First released in Issue 4, Version 2.

25076 **Issue 5**

25077 Moved from X/OPEN UNIX extension to BASE.

25078 **NAME**25079 mktemp — make a unique filename (**LEGACY**)25080 **SYNOPSIS**25081 xSI `#include <stdlib.h>`25082 `char *mktemp(char *template);`

25083

25084 **DESCRIPTION**

25085 The *mktemp()* function shall replace the contents of the string pointed to by *template* by a unique
 25086 filename and return *template*. The application shall initialize *template* to be a filename with six
 25087 trailing 'X's; *mktemp()* shall replace each 'X' with a single byte character from the portable
 25088 filename character set.

25089 **RETURN VALUE**

25090 The *mktemp()* function shall return the pointer *template*. If a unique name cannot be created,
 25091 *template* shall point to a null string.

25092 **ERRORS**

25093 No errors are defined.

25094 **EXAMPLES**25095 **Generating a Filename**

25096 The following example replaces the contents of the "template" string with a 10-character
 25097 filename beginning with the characters "file" and returns a pointer to the "template" string
 25098 that contains the new filename.

25099 `#include <stdlib.h>`25100 `...`25101 `char *template = "/tmp/fileXXXXXX";`25102 `char *ptr;`25103 `ptr = mktemp(template);`25104 **APPLICATION USAGE**

25105 Between the time a pathname is created and the file opened, it is possible for some other process
 25106 to create a file with the same name. The *mkstemp()* function avoids this problem and is preferred
 25107 over this function.

25108 **RATIONALE**

25109 None.

25110 **FUTURE DIRECTIONS**

25111 This function may be withdrawn in a future version.

25112 **SEE ALSO**25113 *mkstemp()*, *tmpfile()*, *tmpnam()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**25114 **CHANGE HISTORY**

25115 First released in Issue 4, Version 2.

25116 **Issue 5**

25117 Moved from X/OPEN UNIX extension to BASE.

25118 **Issue 6**

25119 This function is marked LEGACY.

25120 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25121 **NAME**

25122 mktime — convert broken-down time into time since the Epoch

25123 **SYNOPSIS**

25124 #include <time.h>

25125 time_t mktime(struct tm *timeptr);

25126 **DESCRIPTION**

25127 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 25128 conflict between the requirements described here and the ISO C standard is unintentional. This
 25129 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

25130 The *mktime()* function shall convert the broken-down time, expressed as local time, in the
 25131 structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that
 25132 of the values returned by *time()*. The original values of the *tm_wday* and *tm_yday* components of
 25133 the structure are ignored, and the original values of the other components are not restricted to
 25134 the ranges described in <time.h>.

25135 CX A positive or 0 value for *tm_isdst* shall cause *mktime()* to presume initially that Daylight Savings
 25136 Time, respectively, is or is not in effect for the specified time. A negative value for *tm_isdst* shall
 25137 cause *mktime()* to attempt to determine whether Daylight Savings Time is in effect for the
 25138 specified time.

25139 Local timezone information shall be set as though *mktime()* called *tzset()*.

25140 The relationship between the **tm** structure (defined in the <time.h> header) and the time in
 25141 seconds since the Epoch is that the result shall be as specified in the expression given in the
 25142 definition of seconds since the Epoch (see the Base Definitions volume of IEEE Std 1003.1-2001,
 25143 Section 4.14, Seconds Since the Epoch) corrected for timezone and any seasonal time
 25144 adjustments, where the names in the structure and in the expression correspond.

25145 Upon successful completion, the values of the *tm_wday* and *tm_yday* components of the structure
 25146 shall be set appropriately, and the other components are set to represent the specified time since
 25147 the Epoch, but with their values forced to the ranges indicated in the <time.h> entry; the final
 25148 value of *tm_mday* shall not be set until *tm_mon* and *tm_year* are determined.

25149 **RETURN VALUE**

25150 The *mktime()* function shall return the specified time since the Epoch encoded as a value of type
 25151 **time_t**. If the time since the Epoch cannot be represented, the function shall return the value
 25152 (**time_t**)-1.

25153 **ERRORS**

25154 No errors are defined.

25155 **EXAMPLES**

25156 What day of the week is July 4, 2001?

25157 #include <stdio.h>

25158 #include <time.h>

25159 struct tm time_str;

25160 char daybuf[20];

25161 int main(void)

25162 {

25163 time_str.tm_year = 2001 - 1900;

25164 time_str.tm_mon = 7 - 1;

25165 time_str.tm_mday = 4;

```
25166         time_str.tm_hour = 0;
25167         time_str.tm_min = 0;
25168         time_str.tm_sec = 1;
25169         time_str.tm_isdst = -1;
25170         if (mktime(&time_str) == -1)
25171             (void)puts("-unknown-");
25172         else {
25173             (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
25174             (void)puts(daybuf);
25175         }
25176         return 0;
25177     }
```

25178 APPLICATION USAGE

25179 None.

25180 RATIONALE

25181 None.

25182 FUTURE DIRECTIONS

25183 None.

25184 SEE ALSO

25185 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
25186 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

25187 CHANGE HISTORY

25188 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C
25189 standard.

25190 Issue 6

25191 Extensions beyond the ISO C standard are marked.

25192 **NAME**25193 mlock, munlock — lock or unlock a range of process address space (**REALTIME**)25194 **SYNOPSIS**25195 MLR `#include <sys/mman.h>`25196 `int mlock(const void *addr, size_t len);`25197 `int munlock(const void *addr, size_t len);`

25198

25199 **DESCRIPTION**

25200 The *mlock()* function shall cause those whole pages containing any part of the address space of
 25201 the process starting at address *addr* and continuing for *len* bytes to be memory-resident until
 25202 unlocked or until the process exits or *execs* another process image. The implementation may
 25203 require that *addr* be a multiple of {PAGESIZE}.

25204 The *munlock()* function shall unlock those whole pages containing any part of the address space
 25205 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many
 25206 times *mlock()* has been called by the process for any of the pages in the specified range. The
 25207 implementation may require that *addr* be a multiple of {PAGESIZE}.

25208 If any of the pages in the range specified to a call to *munlock()* are also mapped into the address
 25209 spaces of other processes, any locks established on those pages by another process are
 25210 unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a
 25211 call to *munlock()* are also mapped into other portions of the address space of the calling process
 25212 outside the range specified, any locks established on those pages via the other mappings are also
 25213 unaffected by this call.

25214 Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-
 25215 resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked
 25216 with respect to the address space of the process. Memory residency of unlocked pages is
 25217 unspecified.

25218 The appropriate privilege is required to lock process memory with *mlock()*.

25219 **RETURN VALUE**

25220 Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero.
 25221 Otherwise, no change is made to any locks in the address space of the process, and the function
 25222 shall return a value of -1 and set *errno* to indicate the error.

25223 **ERRORS**

25224 The *mlock()* and *munlock()* functions shall fail if:

25225 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does
 25226 not correspond to valid mapped pages in the address space of the process.

25227 The *mlock()* function shall fail if:

25228 [EAGAIN] Some or all of the memory identified by the operation could not be locked
 25229 when the call was made.

25230 The *mlock()* and *munlock()* functions may fail if:

25231 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

25232 The *mlock()* function may fail if:

25233 [ENOMEM] Locking the pages mapped by the specified range would exceed an
 25234 implementation-defined limit on the amount of memory that the process may
 25235 lock.

25236 [EPERM] The calling process does not have the appropriate privilege to perform the
25237 requested operation.

25238 **EXAMPLES**

25239 None.

25240 **APPLICATION USAGE**

25241 None.

25242 **RATIONALE**

25243 None.

25244 **FUTURE DIRECTIONS**

25245 None.

25246 **SEE ALSO**

25247 *exec*, *exit()*, *fork()*, *mlockall()*, *munmap()*, the Base Definitions volume of IEEE Std 1003.1-2001,
25248 **<sys/mman.h>**

25249 **CHANGE HISTORY**

25250 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25251 **Issue 6**

25252 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.

25253 The [ENOSYS] error condition has been removed as stubs need not be provided if an
25254 implementation does not support the Range Memory Locking option.

25255 **NAME**25256 mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)25257 **SYNOPSIS**25258 ML `#include <sys/mman.h>`25259 `int mlockall(int flags);`25260 `int munlockall(void);`

25261

25262 **DESCRIPTION**

25263 The *mlockall()* function shall cause all of the pages mapped by the address space of a process to
 25264 be memory-resident until unlocked or until the process exits or *execs* another process image. The
 25265 *flags* argument determines whether the pages to be locked are those currently mapped by the
 25266 address space of the process, those that are mapped in the future, or both. The *flags* argument is
 25267 constructed from the bitwise-inclusive OR of one or more of the following symbolic constants,
 25268 defined in **<sys/mman.h>**:

25269 MCL_CURRENT Lock all of the pages currently mapped into the address space of the process.

25270 MCL_FUTURE Lock all of the pages that become mapped into the address space of the
 25271 process in the future, when those mappings are established.

25272 If MCL_FUTURE is specified, and the automatic locking of future mappings eventually causes
 25273 the amount of locked memory to exceed the amount of available physical memory or any other
 25274 implementation-defined limit, the behavior is implementation-defined. The manner in which the
 25275 implementation informs the application of these situations is also implementation-defined.

25276 The *munlockall()* function shall unlock all currently mapped pages of the address space of the
 25277 process. Any pages that become mapped into the address space of the process after a call to
 25278 *munlockall()* shall not be locked, unless there is an intervening call to *mlockall()* specifying
 25279 MCL_FUTURE or a subsequent call to *mlockall()* specifying MCL_CURRENT. If pages mapped
 25280 into the address space of the process are also mapped into the address spaces of other processes
 25281 and are locked by those processes, the locks established by the other processes shall be
 25282 unaffected by a call by this process to *munlockall()*.

25283 Upon successful return from the *mlockall()* function that specifies MCL_CURRENT, all currently
 25284 mapped pages of the process' address space shall be memory-resident and locked. Upon return
 25285 from the *munlockall()* function, all currently mapped pages of the process' address space shall be
 25286 unlocked with respect to the process' address space. The memory residency of unlocked pages is
 25287 unspecified.

25288 The appropriate privilege is required to lock process memory with *mlockall()*.25289 **RETURN VALUE**

25290 Upon successful completion, the *mlockall()* function shall return a value of zero. Otherwise, no
 25291 additional memory shall be locked, and the function shall return a value of -1 and set *errno* to
 25292 indicate the error. The effect of failure of *mlockall()* on previously existing locks in the address
 25293 space is unspecified.

25294 If it is supported by the implementation, the *munlockall()* function shall always return a value of
 25295 zero. Otherwise, the function shall return a value of -1 and set *errno* to indicate the error.

25296 **ERRORS**25297 The *mlockall()* function shall fail if:

25298 [EAGAIN] Some or all of the memory identified by the operation could not be locked
 25299 when the call was made.

25300	[EINVAL]	The <i>flags</i> argument is zero, or includes unimplemented flags.
25301		The <i>mlockall()</i> function may fail if:
25302	[ENOMEM]	Locking all of the pages currently mapped into the address space of the
25303		process would exceed an implementation-defined limit on the amount of
25304		memory that the process may lock.
25305	[EPERM]	The calling process does not have the appropriate privilege to perform the
25306		requested operation.
25307	EXAMPLES	
25308		None.
25309	APPLICATION USAGE	
25310		None.
25311	RATIONALE	
25312		None.
25313	FUTURE DIRECTIONS	
25314		None.
25315	SEE ALSO	
25316		<i>exec</i> , <i>exit()</i> , <i>fork()</i> , <i>mlock()</i> , <i>munmap()</i> , the Base Definitions volume of IEEE Std 1003.1-2001,
25317		< sys/mman.h >
25318	CHANGE HISTORY	
25319		First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
25320	Issue 6	
25321		The <i>mlockall()</i> and <i>munlockall()</i> functions are marked as part of the Process Memory Locking
25322		option.
25323		The [ENOSYS] error condition has been removed as stubs need not be provided if an
25324		implementation does not support the Process Memory Locking option.

25325 **NAME**

25326 mmap — map pages of memory

25327 **SYNOPSIS**

25328 MC3 #include <sys/mman.h>

```
25329 void *mmap(void *addr, size_t len, int prot, int flags,
25330           int fildes, off_t off);
```

25331

25332 **DESCRIPTION**

25333 The *mmap()* function shall establish a mapping between a process' address space and a file,
 25334 TYM shared memory object, or typed memory object. The format of the call is as follows:

```
25335 pa=mmap(addr, len, prot, flags, fildes, off);
```

25336 The *mmap()* function shall establish a mapping between the address space of the process at an
 25337 address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off*
 25338 for *len* bytes. The value of *pa* is an implementation-defined function of the parameter *addr* and
 25339 the values of *flags*, further described below. A successful *mmap()* call shall return *pa* as its result.
 25340 The address range starting at *pa* and continuing for *len* bytes shall be legitimate for the possible
 25341 (not necessarily current) address space of the process. The range of bytes starting at *off* and
 25342 continuing for *len* bytes shall be legitimate for the possible (not necessarily current) offsets in the
 25343 TYM file, shared memory object, or typed memory object represented by *fildes*.

25344 TYM If *fildes* represents a typed memory object opened with either the
 25345 POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG
 25346 flag, the memory object to be mapped shall be that portion of the typed memory object allocated
 25347 by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap()*
 25348 is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling
 25349 process, *mmap()* shall fail.

25350 The mapping established by *mmap()* shall replace any previous mappings for those whole pages
 25351 containing any part of the address space of the process starting at *pa* and continuing for *len*
 25352 bytes.

25353 If the size of the mapped file changes after the call to *mmap()* as a result of some other operation
 25354 on the mapped file, the effect of references to portions of the mapped region that correspond to
 25355 added or removed portions of the file is unspecified.

25356 TYM The *mmap()* function shall be supported for regular files, shared memory objects, and typed
 25357 memory objects. Support for any other type of file is unspecified.

25358 The parameter *prot* determines whether read, write, execute, or some combination of accesses
 25359 are permitted to the data being mapped. The *prot* shall be either PROT_NONE or the bitwise-
 25360 inclusive OR of one or more of the other flags in the following table, defined in the
 25361 <sys/mman.h> header.

25362

25363

25364

25365

25366

25367

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

25368 If an implementation cannot support the combination of access types specified by *prot*, the call
 25369 to *mmap()* shall fail.

25370 MPR An implementation may permit accesses other than those specified by *prot*; however, if the
 25371 Memory Protection option is supported, the implementation shall not permit a write to succeed
 25372 where PROT_WRITE has not been set or shall not permit any access where PROT_NONE alone
 25373 has been set. The implementation shall support at least the following values of *prot*:
 25374 PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-inclusive OR of PROT_READ and
 25375 PROT_WRITE. If the Memory Protection option is not supported, the result of any access that
 25376 conflicts with the specified protection is undefined. The file descriptor *fildes* shall have been
 25377 opened with read permission, regardless of the protection options specified. If PROT_WRITE is
 25378 specified, the application shall ensure that it has opened the file descriptor *fildes* with write
 25379 permission unless MAP_PRIVATE is specified in the *flags* parameter as described below.

25380 The parameter *flags* provides other information about the handling of the mapped data. The
 25381 value of *flags* is the bitwise-inclusive OR of these options, defined in `<sys/mman.h>`:

25382

25383

25384

25385

25386

Symbolic Constant	Description
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret <i>addr</i> exactly.

25387 Implementations that do not support the Memory Mapped Files option are not required to
 25388 support MAP_PRIVATE.

25389 XSI It is implementation-defined whether MAP_FIXED shall be supported. MAP_FIXED shall be
 25390 supported on XSI-conformant systems.

25391 MAP_SHARED and MAP_PRIVATE describe the disposition of write references to the memory
 25392 object. If MAP_SHARED is specified, write references shall change the underlying object. If
 25393 MAP_PRIVATE is specified, modifications to the mapped data by the calling process shall be
 25394 visible only to the calling process and shall not change the underlying object. It is unspecified
 25395 whether modifications to the underlying object done after the MAP_PRIVATE mapping is
 25396 established are visible through the MAP_PRIVATE mapping. Either MAP_SHARED or
 25397 MAP_PRIVATE can be specified, but not both. The mapping type is retained across *fork()*.

25398 TYM When *fildes* represents a typed memory object opened with either the
 25399 POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG
 25400 flag, *mmap()* shall, if there are enough resources available, map *len* bytes allocated from the
 25401 corresponding typed memory object which were not previously allocated to any process in any
 25402 processor that may access that typed memory object. If there are not enough resources available,
 25403 the function shall fail. If *fildes* represents a typed memory object opened with the
 25404 POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, these allocated bytes shall be contiguous
 25405 within the typed memory object. If *fildes* represents a typed memory object opened with the
 25406 POSIX_TYPED_MEM_ALLOCATE flag, these allocated bytes may be composed of non-
 25407 contiguous fragments within the typed memory object. If *fildes* represents a typed memory
 25408 object opened with neither the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag nor the
 25409 POSIX_TYPED_MEM_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory
 25410 object are mapped, exactly as when mapping a file or shared memory object. In this case, if two
 25411 processes map an area of typed memory using the same *off* and *len* values and using file
 25412 descriptors that refer to the same memory pool (either from the same port or from a different
 25413 port), both processes shall map the same region of storage.

25414 When MAP_FIXED is set in the *flags* argument, the implementation is informed that the value of
 25415 *pa* shall be *addr*, exactly. If MAP_FIXED is set, *mmap()* may return MAP_FAILED and set *errno* to
 25416 [EINVAL]. If a MAP_FIXED request is successful, the mapping established by *mmap()* replaces
 25417 any previous mappings for the process' pages in the range [*pa*,*pa+len*).

When MAP_FIXED is not set, the implementation uses *addr* in an implementation-defined manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the implementation deems suitable for a mapping of *len* bytes to the file. All implementations interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*, subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a process address near which the mapping should be placed. When the implementation selects a value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

The *off* argument is constrained to be aligned and sized according to the value returned by *sysconf()* when passed *_SC_PAGESIZE* or *_SC_PAGE_SIZE*. When MAP_FIXED is specified, the application shall ensure that the argument *addr* also meets these constraints. The implementation performs mapping operations over whole pages. Thus, while the argument *len* need not meet a size or alignment constraint, the implementation shall include, in any mapping operation, any partial page specified by the range [*pa*,*pa+len*).

The system shall always zero-fill any partial page at the end of an object. Further, the system shall never write out any modified portions of the last page of an object which are beyond its end. References within the address range starting at *pa* and continuing for *len* bytes to whole pages following the end of an object shall result in delivery of a SIGBUS signal.

An implementation may generate SIGBUS signals when a reference would cause an error in the mapped object, such as out-of-space condition.

The *mmap()* function shall add an extra reference to the file associated with the file descriptor *fd* which is not removed by a subsequent *close()* on that file descriptor. This reference shall be removed when there are no more mappings to the file.

The *st_atime* field of the mapped file may be marked for update at any time between the *mmap()* call and the corresponding *munmap()* call. The initial read or write reference to a mapped region shall cause the file's *st_atime* field to be marked for update if it has not already been marked for update.

The *st_ctime* and *st_mtime* fields of a file that is mapped with MAP_SHARED and PROT_WRITE shall be marked for update at some point in the interval between a write reference to the mapped region and the next call to *msync()* with MS_ASYNC or MS_SYNC for that portion of the file by any process. If there is no such call and if the underlying file is modified as a result of a write reference, then these fields shall be marked for update at some time after the write reference.

There may be implementation-defined limits on the number of memory regions that can be mapped (per process or per system).

If such a limit is imposed, whether the number of memory regions that can be mapped by a process is decreased by the use of *shmat()* is implementation-defined.

If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the mappings in the address range starting at *addr* and continuing for *len* bytes may have been unmapped.

25457 RETURN VALUE

Upon successful completion, the *mmap()* function shall return the address at which the mapping was placed (*pa*); otherwise, it shall return a value of MAP_FAILED and set *errno* to indicate the error. The symbol MAP_FAILED is defined in the <sys/mman.h> header. No successful return from *mmap()* shall return the value MAP_FAILED.

25462 **ERRORS**25463 The *mmap()* function shall fail if:

25464 [EACCES] The *fil*des argument is not open for read, regardless of the protection specified,
 25465 or *fil*des is not open for write and PROT_WRITE was specified for a
 25466 MAP_SHARED type mapping.

25467 ML [EAGAIN] The mapping could not be locked in memory, if required by *mlockall()*, due to
 25468 a lack of resources.

25469 [EBADF] The *fil*des argument is not a valid open file descriptor.

25470 [EINVAL] The *addr* argument (if MAP_FIXED was specified) or *off* is not a multiple of
 25471 the page size as returned by *sysconf()*, or is considered invalid by the
 25472 implementation.

25473 [EINVAL] The value of *flags* is invalid (neither MAP_PRIVATE nor MAP_SHARED is
 25474 set).

25475 [EMFILE] The number of mapped regions would exceed an implementation-defined
 25476 limit (per process or per system).

25477 [ENODEV] The *fil*des argument refers to a file whose type is not supported by *mmap()*.

25478 [ENOMEM] MAP_FIXED was specified, and the range [*addr*,*addr+len*) exceeds that allowed
 25479 for the address space of a process; or, if MAP_FIXED was not specified and
 25480 there is insufficient room in the address space to effect the mapping.

25481 ML [ENOMEM] The mapping could not be locked in memory, if required by *mlockall()*,
 25482 because it would require more space than the system is able to supply.

25483 TYM [ENOMEM] Not enough unallocated memory resources remain in the typed memory
 25484 object designated by *fil*des to allocate *len* bytes.

25485 [ENOTSUP] MAP_FIXED or MAP_PRIVATE was specified in the *flags* argument and the
 25486 implementation does not support this functionality.

25487 The implementation does not support the combination of accesses requested
 25488 in the *prot* argument.

25489 [ENXIO] Addresses in the range [*off*,*off+len*) are invalid for the object specified by *fil*des.

25490 [ENXIO] MAP_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is
 25491 invalid for the object specified by *fil*des.

25492 TYM [ENXIO] The *fil*des argument refers to a typed memory object that is not accessible from
 25493 the calling process.

25494 [EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset
 25495 maximum established in the open file description associated with *fil*des.

25496 **EXAMPLES**

25497 None.

25498 **APPLICATION USAGE**

25499 Use of *mmap()* may reduce the amount of memory available to other memory allocation
 25500 functions.

25501 Use of MAP_FIXED may result in unspecified behavior in further use of *malloc()* and *shmat()*.
 25502 The use of MAP_FIXED is discouraged, as it may prevent an implementation from making the
 25503 most effective use of resources.

25504 The application must ensure correct synchronization when using *mmap()* in conjunction with
 25505 any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

25506 The *mmap()* function allows access to resources via address space manipulations, instead of
 25507 *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the
 25508 address to which the file was mapped. So, using pseudo-code to illustrate the way in which an
 25509 existing program might be changed to use *mmap()*, the following:

```
25510 fildes = open(...)
25511 lseek(fildes, some_offset)
25512 read(fildes, buf, len)
25513 /* Use data in buf. */
```

25514 becomes:

```
25515 fildes = open(...)
25516 address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
25517 /* Use data at address. */
```

25518 RATIONALE

25519 After considering several other alternatives, it was decided to adopt the *mmap()* definition found
 25520 in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is
 25521 minimal, in that it describes only what has been built, and what appears to be necessary for a
 25522 general and portable mapping facility.

25523 Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose
 25524 mapping facility. It can be used to map any appropriate object, such as memory, files, devices,
 25525 and so on, into the address space of a process.

25526 When a mapping is established, it is possible that the implementation may need to map more
 25527 than is requested into the address space of the process because of hardware requirements. An
 25528 application, however, cannot count on this behavior. Implementations that do not use a paged
 25529 architecture may simply allocate a common memory region and return the address of it; such
 25530 implementations probably do not allocate any more than is necessary. References past the end of
 25531 the requested area are unspecified.

25532 If an application requests a mapping that would overlay existing mappings in the process, it
 25533 might be desirable that an implementation detect this and inform the application. However, the
 25534 default, portable (not *MAP_FIXED*) operation does not overlay existing mappings. On the other
 25535 hand, if the program specifies a fixed address mapping (which requires some implementation
 25536 knowledge to determine a suitable address, if the function is supported at all), then the program
 25537 is presumed to be successfully managing its own address space and should be trusted when it
 25538 asks to map over existing data structures. Furthermore, it is also desirable to make as few system
 25539 calls as possible, and it might be considered onerous to require an *munmap()* before an *mmap()*
 25540 to the same address range. This volume of IEEE Std 1003.1-2001 specifies that the new mappings
 25541 replace any existing mappings, following existing practice in this regard.

25542 It is not expected, when the Memory Protection option is supported, that all hardware
 25543 implementations are able to support all combinations of permissions at all addresses. When this
 25544 option is supported, implementations are required to disallow write access to mappings without
 25545 write permission and to disallow access to mappings without any access permission. Other than
 25546 these restrictions, implementations may allow access types other than those requested by the
 25547 application. For example, if the application requests only *PROT_WRITE*, the implementation
 25548 may also allow read access. A call to *mmap()* fails if the implementation cannot support allowing
 25549 all the access requested by the application. For example, some implementations cannot support
 25550 a request for both write access and execute access simultaneously. All implementations
 25551 supporting the Memory Protection option must support requests for no access, read access,

25552 write access, and both read and write access. Strictly conforming code must only rely on the
25553 required checks. These restrictions allow for portability across a wide range of hardware.

25554 The MAP_FIXED address treatment is likely to fail for non-page-aligned values and for certain
25555 architecture-dependent address ranges. Conforming implementations cannot count on being
25556 able to choose address values for MAP_FIXED without utilizing non-portable, implementation-
25557 defined knowledge. Nonetheless, MAP_FIXED is provided as a standard interface conforming to
25558 existing practice for utilizing such knowledge when it is available.

25559 Similarly, in order to allow implementations that do not support virtual addresses, support for
25560 directly specifying any mapping addresses via MAP_FIXED is not required and thus a
25561 conforming application may not count on it.

25562 The MAP_PRIVATE function can be implemented efficiently when memory protection hardware
25563 is available. When such hardware is not available, implementations can implement such
25564 “mappings” by simply making a real copy of the relevant data into process private memory,
25565 though this tends to behave similarly to *read()*.

25566 The function has been defined to allow for many different models of using shared memory.
25567 However, all uses are not equally portable across all machine architectures. In particular, the
25568 *mmap()* function allows the system as well as the application to specify the address at which to
25569 map a specific region of a memory object. The most portable way to use the function is always to
25570 let the system choose the address, specifying NULL as the value for the argument *addr* and not
25571 to specify MAP_FIXED.

25572 If it is intended that a particular region of a memory object be mapped at the same address in a
25573 group of processes (on machines where this is even possible), then MAP_FIXED can be used to
25574 pass in the desired mapping address. The system can still be used to choose the desired address
25575 if the first such mapping is made without specifying MAP_FIXED, and then the resulting
25576 mapping address can be passed to subsequent processes for them to pass in via MAP_FIXED.
25577 The availability of a specific address range cannot be guaranteed, in general.

25578 The *mmap()* function can be used to map a region of memory that is larger than the current size
25579 of the object. Memory access within the mapping but beyond the current end of the underlying
25580 objects may result in SIGBUS signals being sent to the process. The reason for this is that the size
25581 of the object can be manipulated by other processes and can change at any moment. The
25582 implementation should tell the application that a memory reference is outside the object where
25583 this can be detected; otherwise, written data may be lost and read data may not reflect actual
25584 data in the object.

25585 Note that references beyond the end of the object do not extend the object as the new end cannot
25586 be determined precisely by most virtual memory hardware. Instead, the size can be directly
25587 manipulated by *ftruncate()*.

25588 Process memory locking does apply to shared memory regions, and the MEMLOCK_FUTURE
25589 argument to *mlockall()* can be relied upon to cause new shared memory regions to be
25590 automatically locked.

25591 Existing implementations of *mmap()* return the value -1 when unsuccessful. Since the casting of
25592 this value to type **void *** cannot be guaranteed by the ISO C standard to be distinct from a
25593 successful value, this volume of IEEE Std 1003.1-2001 defines the symbol MAP_FAILED, which a
25594 conforming implementation does not return as the result of a successful call.

25595 **FUTURE DIRECTIONS**

25596 None.

25597 **SEE ALSO**

25598 *exec*, *fcntl()*, *fork()*, *lockf()*, *msync()*, *munmap()*, *mprotect()*, *posix_typed_mem_open()*, *shmat()*,
 25599 *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/mman.h>

25600 **CHANGE HISTORY**

25601 First released in Issue 4, Version 2.

25602 **Issue 5**

25603 Moved from X/OPEN UNIX extension to BASE.

25604 Aligned with *mmap()* in the POSIX Realtime Extension as follows:

- 25605 • The DESCRIPTION is extensively reworded.
- 25606 • The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- 25607 • New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- 25608 • The value returned on failure is the value of the constant MAP_FAILED; this was previously
- 25609 defined as -1.

25610 Large File Summit extensions are added.

25611 **Issue 6**

25612 The *mmap()* function is marked as part of the Memory Mapped Files option.

25613 The Open Group Corrigendum U028/6 is applied, changing (void *)-1 to MAP_FAILED.

25614 The following new requirements on POSIX implementations derive from alignment with the
 25615 Single UNIX Specification:

- 25616 • The DESCRIPTION is updated to describe the use of MAP_FIXED.
- 25617 • The DESCRIPTION is updated to describe the addition of an extra reference to the file
- 25618 associated with the file descriptor passed to *mmap()*.
- 25619 • The DESCRIPTION is updated to state that there may be implementation-defined limits on
- 25620 the number of memory regions that can be mapped.
- 25621 • The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*
- 25622 argument.
- 25623 • The [EINVAL] and [EMFILE] error conditions are added.
- 25624 • The [EOVERFLOW] error condition is added. This change is to support large files.

25625 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 25626 • The DESCRIPTION is updated to describe the cases when MAP_PRIVATE and MAP_FIXED
- 25627 need not be supported.

25628 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 25629 • Semantics for typed memory objects are added to the DESCRIPTION.
- 25630 • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 25631 • The *posix_typed_mem_open()* function is added to the SEE ALSO section.

25632 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25633 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code |
 25634 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM). |

25635 **NAME**

25636 modf, modff, modfl — decompose a floating-point number

25637 **SYNOPSIS**

25638 #include <math.h>

25639 double modf(double *x*, double **iptr*);25640 float modff(float *value*, float **iptr*);25641 long double modfl(long double *value*, long double **iptr*);25642 **DESCRIPTION**

25643 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 25644 conflict between the requirements described here and the ISO C standard is unintentional. This
 25645 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

25646 These functions shall break the argument *x* into integral and fractional parts, each of which has
 25647 the same sign as the argument. It stores the integral part as a **double** (for the *modf*() function), a
 25648 **float** (for the *modff*() function), or a **long double** (for the *modfl*() function), in the object pointed
 25649 to by *iptr*.

25650 **RETURN VALUE**25651 Upon successful completion, these functions shall return the signed fractional part of *x*.25652 MX If *x* is NaN, a NaN shall be returned, and **iptr* shall be set to a NaN.25653 If *x* is $\pm\text{Inf}$, ± 0 shall be returned, and **iptr* shall be set to $\pm\text{Inf}$.25654 **ERRORS**

25655 No errors are defined.

25656 **EXAMPLES**

25657 None.

25658 **APPLICATION USAGE**25659 The *modf*() function computes the function result and **iptr* such that:25660 *a* = modf(*x*, *iptr*) ;25661 *x* == *a*+**iptr* ;

25662 allowing for the usual floating-point inaccuracies.

25663 **RATIONALE**

25664 None.

25665 **FUTURE DIRECTIONS**

25666 None.

25667 **SEE ALSO**25668 *frexp*(), *isnan*(), *ldexp*(), the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>25669 **CHANGE HISTORY**

25670 First released in Issue 1. Derived from Issue 1 of the SVID.

25671 **Issue 5**

25672 The DESCRIPTION is updated to indicate how an application should check for an error. This
 25673 text was previously published in the APPLICATION USAGE section.

25674 **Issue 6**

25675 The *modff*() and *modfl*() functions are added for alignment with the ISO/IEC 9899:1999
 25676 standard.

25677	The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
25678	revised to align with the ISO/IEC 9899:1999 standard.
25679	IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
25680	marked.
25681	IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example
25682	in the APPLICATION USAGE section.

25683 **NAME**

25684 mprotect — set protection of memory mapping

25685 **SYNOPSIS**

25686 MPR #include <sys/mman.h>

25687 int mprotect(void *addr, size_t len, int prot);

25688

25689 **DESCRIPTION**

25690 The *mprotect()* function shall change the access protections to be that specified by *prot* for those
 25691 whole pages containing any part of the address space of the process starting at address *addr* and
 25692 continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some
 25693 combination of accesses are permitted to the data being mapped. The *prot* argument should be
 25694 either PROT_NONE or the bitwise-inclusive OR of one or more of PROT_READ, PROT_WRITE,
 25695 and PROT_EXEC.

25696 If an implementation cannot support the combination of access types specified by *prot*, the call
 25697 to *mprotect()* shall fail.

25698 An implementation may permit accesses other than those specified by *prot*; however, no
 25699 implementation shall permit a write to succeed where PROT_WRITE has not been set or shall
 25700 permit any access where PROT_NONE alone has been set. Implementations shall support at
 25701 least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-
 25702 inclusive OR of PROT_READ and PROT_WRITE. If PROT_WRITE is specified, the application
 25703 shall ensure that it has opened the mapped objects in the specified address range with write
 25704 permission, unless MAP_PRIVATE was specified in the original mapping, regardless of whether
 25705 the file descriptors used to map the objects have since been closed.

25706 The implementation shall require that *addr* be a multiple of the page size as returned by
 25707 sysconf().

25708 The behavior of this function is unspecified if the mapping was not established by a call to
 25709 mmap().

25710 When *mprotect()* fails for reasons other than [EINVAL], the protections on some of the pages in
 25711 the range [*addr*,*addr*+*len*) may have been changed.

25712 **RETURN VALUE**

25713 Upon successful completion, *mprotect()* shall return 0; otherwise, it shall return -1 and set *errno*
 25714 to indicate the error.

25715 **ERRORS**25716 The *mprotect()* function shall fail if:

25717 [EACCES] The *prot* argument specifies a protection that violates the access permission
 25718 the process has to the underlying memory object.

25719 [EAGAIN] The *prot* argument specifies PROT_WRITE over a MAP_PRIVATE mapping
 25720 and there are insufficient memory resources to reserve for locking the private
 25721 page.

25722 [EINVAL] The *addr* argument is not a multiple of the page size as returned by sysconf().

25723 [ENOMEM] Addresses in the range [*addr*,*addr*+*len*) are invalid for the address space of a
 25724 process, or specify one or more pages which are not mapped.

25725 [ENOMEM] The *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and
 25726 it would require more space than the system is able to supply for locking the
 25727 private pages, if required.

25728 [ENOTSUP] The implementation does not support the combination of accesses requested
 25729 in the *prot* argument.

25730 **EXAMPLES**

25731 None.

25732 **APPLICATION USAGE**

25733 The [EINVAL] error above is marked EX because it is defined as an optional error in the POSIX
 25734 Realtime Extension.

25735 **RATIONALE**

25736 None.

25737 **FUTURE DIRECTIONS**

25738 None.

25739 **SEE ALSO**

25740 *mmap()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/mman.h>

25741 **CHANGE HISTORY**

25742 First released in Issue 4, Version 2.

25743 **Issue 5**

25744 Moved from X/OPEN UNIX extension to BASE.

25745 Aligned with *mprotect()* in the POSIX Realtime Extension as follows:

- 25746 • The DESCRIPTION is largely reworded.
- 25747 • [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
- 25748 • [EAGAIN] is moved from the optional to the mandatory error conditions.

25749 **Issue 6**

25750 The *mprotect()* function is marked as part of the Memory Protection option.

25751 The following new requirements on POSIX implementations derive from alignment with the
 25752 Single UNIX Specification:

- 25753 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of
 25754 the page size as returned by *sysconf()*.
- 25755 • The [EINVAL] error condition is added.

25756 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25757 **NAME**25758 mq_close — close a message queue (**REALTIME**)25759 **SYNOPSIS**

25760 MSG #include <mqueue.h>

25761 int mq_close(mqd_t mqdes);

25762

25763 **DESCRIPTION**

25764 The *mq_close()* function shall remove the association between the message queue descriptor,
 25765 *mqdes*, and its message queue. The results of using this message queue descriptor after
 25766 successful return from this *mq_close()*, and until the return of this message queue descriptor
 25767 from a subsequent *mq_open()*, are undefined.

25768 If the process has successfully attached a notification request to the message queue via this
 25769 *mqdes*, this attachment shall be removed, and the message queue is available for another process
 25770 to attach for notification.

25771 **RETURN VALUE**

25772 Upon successful completion, the *mq_close()* function shall return a value of zero; otherwise, the
 25773 function shall return a value of -1 and set *errno* to indicate the error.

25774 **ERRORS**25775 The *mq_close()* function shall fail if:

25776 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

25777 **EXAMPLES**

25778 None.

25779 **APPLICATION USAGE**

25780 None.

25781 **RATIONALE**

25782 None.

25783 **FUTURE DIRECTIONS**

25784 None.

25785 **SEE ALSO**

25786 *mq_open()*, *mq_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of
 25787 IEEE Std 1003.1-2001, <**mqueue.h**>

25788 **CHANGE HISTORY**

25789 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25790 **Issue 6**25791 The *mq_close()* function is marked as part of the Message Passing option.

25792 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 25793 implementation does not support the Message Passing option.

25794 **NAME**25795 mq_getattr — get message queue attributes (**REALTIME**)25796 **SYNOPSIS**

25797 MSG #include <mqueue.h>

25798 int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);

25799

25800 **DESCRIPTION**25801 The *mq_getattr()* function shall obtain status information and attributes of the message queue
25802 and the open message queue description associated with the message queue descriptor.25803 The *mqdes* argument specifies a message queue descriptor.25804 The results shall be returned in the **mq_attr** structure referenced by the *mqstat* argument.25805 Upon return, the following members shall have the values associated with the open message
25806 queue description as set when the message queue was opened and as modified by subsequent
25807 *mq_setattr()* calls: *mq_flags*.25808 The following attributes of the message queue shall be returned as set at message queue
25809 creation: *mq_maxmsg*, *mq_msgsize*.25810 Upon return, the following members within the **mq_attr** structure referenced by the *mqstat*
25811 argument shall be set to the current state of the message queue:25812 *mq_curmsgs* The number of messages currently on the queue.25813 **RETURN VALUE**25814 Upon successful completion, the *mq_getattr()* function shall return zero. Otherwise, the function
25815 shall return -1 and set *errno* to indicate the error.25816 **ERRORS**25817 The *mq_getattr()* function shall fail if:25818 [EBADF] The *mqdes* argument is not a valid message queue descriptor.25819 **EXAMPLES**

25820 None.

25821 **APPLICATION USAGE**

25822 None.

25823 **RATIONALE**

25824 None.

25825 **FUTURE DIRECTIONS**

25826 None.

25827 **SEE ALSO**25828 *mq_open()*, *mq_send()*, *mq_setattr()*, *mq_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the
25829 Base Definitions volume of IEEE Std 1003.1-2001, <mqueue.h>25830 **CHANGE HISTORY**

25831 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25832 **Issue 6**25833 The *mq_getattr()* function is marked as part of the Message Passing option.25834 The [ENOSYS] error condition has been removed as stubs need not be provided if an
25835 implementation does not support the Message Passing option.

25836 The *mq_timedsend()* function is added to the SEE ALSO section for alignment with
25837 IEEE Std 1003.1d-1999.

25838 **NAME**25839 mq_notify — notify process that a message is available (**REALTIME**)25840 **SYNOPSIS**

25841 MSG #include <mqueue.h>

25842 int mq_notify(mqd_t mqdes, const struct sigevent *notification);

25843

25844 **DESCRIPTION**

25845 If the argument *notification* is not NULL, this function shall register the calling process to be
 25846 notified of message arrival at an empty message queue associated with the specified message
 25847 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to
 25848 the process when the message queue transitions from empty to non-empty. At any time, only
 25849 one process may be registered for notification by a message queue. If the calling process or any
 25850 other process has already registered for notification of message arrival at the specified message
 25851 queue, subsequent attempts to register for that message queue shall fail.

25852 If *notification* is NULL and the process is currently registered for notification by the specified
 25853 message queue, the existing registration shall be removed.

25854 When the notification is sent to the registered process, its registration shall be removed. The
 25855 message queue shall then be available for registration.

25856 If a process has registered for notification of message arrival at a message queue and some
 25857 thread is blocked in *mq_receive()* waiting to receive a message when a message arrives at the
 25858 queue, the arriving message shall satisfy the appropriate *mq_receive()*. The resulting behavior is
 25859 as if the message queue remains empty, and no notification shall be sent.

25860 **RETURN VALUE**

25861 Upon successful completion, the *mq_notify()* function shall return a value of zero; otherwise, the
 25862 function shall return a value of -1 and set *errno* to indicate the error.

25863 **ERRORS**25864 The *mq_notify()* function shall fail if:25865 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

25866 [EBUSY] A process is already registered for notification by the message queue.

25867 **EXAMPLES**

25868 None.

25869 **APPLICATION USAGE**

25870 None.

25871 **RATIONALE**

25872 None.

25873 **FUTURE DIRECTIONS**

25874 None.

25875 **SEE ALSO**

25876 *mq_open()*, *mq_send()*, *mq_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base
 25877 Definitions volume of IEEE Std 1003.1-2001, <mqueue.h>

25878 **CHANGE HISTORY**

25879 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25880 **Issue 6**

25881 The *mq_notify()* function is marked as part of the Message Passing option.

25882 The [ENOSYS] error condition has been removed as stubs need not be provided if an
25883 implementation does not support the Message Passing option.

25884 The *mq_timedsend()* function is added to the SEE ALSO section for alignment with
25885 IEEE Std 1003.1d-1999.

25886 **NAME**25887 mq_open — open a message queue (**REALTIME**)25888 **SYNOPSIS**

25889 MSG #include <mqueue.h>

25890 mqd_t mq_open(const char *name, int oflag, ...);

25891

25892 **DESCRIPTION**

25893 The *mq_open()* function shall establish the connection between a process and a message queue
 25894 with a message queue descriptor. It shall create an open message queue description that refers to
 25895 the message queue, and a message queue descriptor that refers to that open message queue
 25896 description. The message queue descriptor is used by other functions to refer to that message
 25897 queue. The *name* argument points to a string naming a message queue. It is unspecified whether
 25898 the name appears in the file system and is visible to other functions that take pathnames as
 25899 arguments. The *name* argument shall conform to the construction rules for a pathname. If *name*
 25900 begins with the slash character, then processes calling *mq_open()* with the same value of *name*
 25901 shall refer to the same message queue object, as long as that name has not been removed. If *name*
 25902 does not begin with the slash character, the effect is implementation-defined. The interpretation
 25903 of slash characters other than the leading slash character in *name* is implementation-defined. If
 25904 the *name* argument is not the name of an existing message queue and creation is not requested,
 25905 *mq_open()* shall fail and return an error.

25906 A message queue descriptor may be implemented using a file descriptor, in which case
 25907 applications can open up to at least {OPEN_MAX} file and message queues.

25908 The *oflag* argument requests the desired receive and/or send access to the message queue. The
 25909 requested access permission to receive messages or send messages shall be granted if the calling
 25910 process would be granted read or write access, respectively, to an equivalently protected file.

25911 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications
 25912 shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

25913 O_RDONLY Open the message queue for receiving messages. The process can use the
 25914 returned message queue descriptor with *mq_receive()*, but not *mq_send()*. A
 25915 message queue may be open multiple times in the same or different processes
 25916 for receiving messages.

25917 O_WRONLY Open the queue for sending messages. The process can use the returned
 25918 message queue descriptor with *mq_send()* but not *mq_receive()*. A message
 25919 queue may be open multiple times in the same or different processes for
 25920 sending messages.

25921 O_RDWR Open the queue for both receiving and sending messages. The process can use
 25922 any of the functions allowed for O_RDONLY and O_WRONLY. A message
 25923 queue may be open multiple times in the same or different processes for
 25924 sending messages.

25925 Any combination of the remaining flags may be specified in the value of *oflag*:

25926 O_CREAT Create a message queue. It requires two additional arguments: *mode*, which
 25927 shall be of type **mode_t**, and *attr*, which shall be a pointer to an **mq_attr**
 25928 structure. If the pathname *name* has already been used to create a message
 25929 queue that still exists, then this flag shall have no effect, except as noted under
 25930 O_EXCL. Otherwise, a message queue shall be created without any messages
 25931 in it. The user ID of the message queue shall be set to the effective user ID of
 25932 the process, and the group ID of the message queue shall be set to the effective

25933		group ID of the process. The file permission bits shall be set to the value of
25934		<i>mode</i> . When bits in <i>mode</i> other than file permission bits are set, the effect is
25935		implementation-defined. If <i>attr</i> is NULL, the message queue shall be created
25936		with implementation-defined default message queue attributes. If <i>attr</i> is non-
25937		NULL and the calling process has the appropriate privilege on <i>name</i> , the
25938		message queue <i>mq_maxmsg</i> and <i>mq_msgsize</i> attributes shall be set to the values
25939		of the corresponding members in the mq_attr structure referred to by <i>attr</i> . If
25940		<i>attr</i> is non-NULL, but the calling process does not have the appropriate
25941		privilege on <i>name</i> , the <i>mq_open()</i> function shall fail and return an error
25942		without creating the message queue.
25943	O_EXCL	If O_EXCL and O_CREAT are set, <i>mq_open()</i> shall fail if the message queue
25944		<i>name</i> exists. The check for the existence of the message queue and the creation
25945		of the message queue if it does not exist shall be atomic with respect to other
25946		threads executing <i>mq_open()</i> naming the same <i>name</i> with O_EXCL and
25947		O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is
25948		undefined.
25949	O_NONBLOCK	Determines whether an <i>mq_send()</i> or <i>mq_receive()</i> waits for resources or
25950		messages that are not currently available, or fails with <i>errno</i> set to [EAGAIN];
25951		see <i>mq_send()</i> and <i>mq_receive()</i> for details.
25952	The <i>mq_open()</i> function does not add or remove messages from the queue.	
25953	RETURN VALUE	
25954	Upon successful completion, the function shall return a message queue descriptor; otherwise,	
25955	the function shall return (mqd_t)−1 and set <i>errno</i> to indicate the error.	
25956	ERRORS	
25957	The <i>mq_open()</i> function shall fail if:	
25958	[EACCES]	The message queue exists and the permissions specified by <i>oflag</i> are denied, or
25959		the message queue does not exist and permission to create the message queue
25960		is denied.
25961	[EEXIST]	O_CREAT and O_EXCL are set and the named message queue already exists.
25962	[EINTR]	The <i>mq_open()</i> function was interrupted by a signal.
25963	[EINVAL]	The <i>mq_open()</i> function is not supported for the given name.
25964	[EINVAL]	O_CREAT was specified in <i>oflag</i> , the value of <i>attr</i> is not NULL, and either
25965		<i>mq_maxmsg</i> or <i>mq_msgsize</i> was less than or equal to zero.
25966	[EMFILE]	Too many message queue descriptors or file descriptors are currently in use by
25967		this process.
25968	[ENAMETOOLONG]	
25969		The length of the <i>name</i> argument exceeds {PATH_MAX} or a pathname
25970		component is longer than {NAME_MAX}.
25971	[ENFILE]	Too many message queues are currently open in the system.
25972	[ENOENT]	O_CREAT is not set and the named message queue does not exist.
25973	[ENOSPC]	There is insufficient space for the creation of the new message queue.

25974 EXAMPLES

25975 None.

25976 APPLICATION USAGE

25977 None.

25978 RATIONALE

25979 None.

25980 FUTURE DIRECTIONS

25981 None.

25982 SEE ALSO

25983 *mq_close()*, *mq_getattr()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_timedreceive()*, *mq_timedsend()*,
25984 *mq_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of
25985 IEEE Std 1003.1-2001, <**mqqueue.h**>

25986 CHANGE HISTORY

25987 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25988 Issue 6

25989 The *mq_open()* function is marked as part of the Message Passing option.

25990 The [ENOSYS] error condition has been removed as stubs need not be provided if an
25991 implementation does not support the Message Passing option.

25992 The *mq_timedreceive()* and *mq_timedsend()* functions are added to the SEE ALSO section for
25993 alignment with IEEE Std 1003.1d-1999.

25994 The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

25995 **NAME**25996 mq_receive, mq_timedreceive — receive a message from a message queue (**REALTIME**)25997 **SYNOPSIS**

25998 MSG #include <mqueue.h>

25999 ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
26000 unsigned *msg_prio);

26001

26002 MSG TMO #include <mqueue.h>

26003 #include <time.h>

26004 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
26005 size_t msg_len, unsigned *restrict msg_prio,
26006 const struct timespec *restrict abs_timeout);

26007

26008 **DESCRIPTION**26009 The *mq_receive()* function shall receive the oldest of the highest priority message(s) from the
26010 message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg_len*
26011 argument, is less than the *mq_msgsize* attribute of the message queue, the function shall fail and
26012 return an error. Otherwise, the selected message shall be removed from the queue and copied to
26013 the buffer pointed to by the *msg_ptr* argument.26014 If the value of *msg_len* is greater than {SSIZE_MAX}, the result is implementation-defined.26015 If the argument *msg_prio* is not NULL, the priority of the selected message shall be stored in the
26016 location referenced by *msg_prio*.26017 If the specified message queue is empty and O_NONBLOCK is not set in the message queue
26018 description associated with *mqdes*, *mq_receive()* shall block until a message is enqueued on the
26019 message queue or until *mq_receive()* is interrupted by a signal. If more than one thread is waiting
26020 to receive a message when a message arrives at an empty queue and the Priority Scheduling
26021 option is supported, then the thread of highest priority that has been waiting the longest shall be
26022 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the
26023 message. If the specified message queue is empty and O_NONBLOCK is set in the message
26024 queue description associated with *mqdes*, no message shall be removed from the queue, and
26025 *mq_receive()* shall return an error.26026 TMO The *mq_timedreceive()* function shall receive the oldest of the highest priority messages from the
26027 message queue specified by *mqdes* as described for the *mq_receive()* function. However, if
26028 O_NONBLOCK was not specified when the message queue was opened via the *mq_open()*
26029 function, and no message exists on the queue to satisfy the receive, the wait for such a message
26030 shall be terminated when the specified timeout expires. If O_NONBLOCK is set, this function is
26031 equivalent to *mq_receive()*.26032 The timeout expires when the absolute time specified by *abs_timeout* passes, as measured by the
26033 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
26034 *abs_timeout*), or if the absolute time specified by *abs_timeout* has already been passed at the time
26035 of the call.26036 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock; if
26037 the Timers option is not supported, the timeout shall be based on the system clock as returned
26038 by the *time()* function.26039 TMO The resolution of the timeout shall be the resolution of the clock on which it is based. The
26040 *timespec* argument is defined in the <time.h> header.

26041 Under no circumstance shall the operation fail with a timeout if a message can be removed from
 26042 the message queue immediately. The validity of the *abs_timeout* parameter need not be checked
 26043 if a message can be removed from the message queue immediately.

26044 RETURN VALUE

26045 TMO Upon successful completion, the *mq_receive()* and *mq_timedreceive()* functions shall return the
 26046 length of the selected message in bytes and the message shall be removed from the queue.
 26047 Otherwise, no message shall be removed from the queue, the functions shall return a value of -1,
 26048 and set *errno* to indicate the error.

26049 ERRORS

26050 TMO The *mq_receive()* and *mq_timedreceive()* functions shall fail if:

26051 [EAGAIN] O_NONBLOCK was set in the message description associated with *mqdes*,
 26052 and the specified message queue is empty.

26053 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.

26054 [EMSGSIZE] The specified message buffer size, *msg_len*, is less than the message size
 26055 attribute of the message queue.

26056 TMO [EINTR] The *mq_receive()* or *mq_timedreceive()* operation was interrupted by a signal.

26057 TMO [EINVAL] The process or thread would have blocked, and the *abs_timeout* parameter
 26058 specified a nanoseconds field value less than zero or greater than or equal to
 26059 1 000 million.

26060 TMO [ETIMEDOUT] The O_NONBLOCK flag was not set when the message queue was opened,
 26061 but no message arrived on the queue before the specified timeout expired.

26062 TMO The *mq_receive()* and *mq_timedreceive()* functions may fail if:

26063 [EBADMSG] The implementation has detected a data corruption problem with the
 26064 message.

26065 EXAMPLES

26066 None.

26067 APPLICATION USAGE

26068 None.

26069 RATIONALE

26070 None.

26071 FUTURE DIRECTIONS

26072 None.

26073 SEE ALSO

26074 *mq_open()*, *mq_send()*, *mq_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *time()*, the Base
 26075 Definitions volume of IEEE Std 1003.1-2001, <mqqueue.h>, <time.h>

26076 CHANGE HISTORY

26077 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26078 Issue 6

26079 The *mq_receive()* function is marked as part of the Message Passing option.

26080 The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to
 26081 *msg_len* rather than *maxsize*.

26082 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 26083 implementation does not support the Message Passing option.

- 26084 The following new requirements on POSIX implementations derive from alignment with the
26085 Single UNIX Specification:
- 26086 • In this function it is possible for the return value to exceed the range of the type **ssize_t** (since
26087 **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of
26088 the **size_t** object is added to the description to resolve this conflict.
- 26089 The *mq_timedreceive()* function is added for alignment with IEEE Std 1003.1d-1999.
- 26090 The **restrict** keyword is added to the *mq_timedreceive()* prototype for alignment with the
26091 ISO/IEC 9899:1999 standard.
- 26092 IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq_timedreceive()*
26093 from **int** to **ssize_t**.

26094 **NAME**26095 mq_send, mq_timedsend — send a message to a message queue (**REALTIME**)26096 **SYNOPSIS**

26097 MSG #include <mqueue.h>

```
26098 int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
26099             unsigned msg_prio);
```

26100

26101 MSG TMO #include <mqueue.h>

26102 #include <time.h>

```
26103 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
26104                  unsigned msg_prio, const struct timespec *abs_timeout);
```

26105

26106 **DESCRIPTION**

26107 The *mq_send()* function shall add the message pointed to by the argument *msg_ptr* to the
 26108 message queue specified by *mqdes*. The *msg_len* argument specifies the length of the message, in
 26109 bytes, pointed to by *msg_ptr*. The value of *msg_len* shall be less than or equal to the *mq_msgsize*
 26110 attribute of the message queue, or *mq_send()* shall fail.

26111 If the specified message queue is not full, *mq_send()* shall behave as if the message is inserted
 26112 into the message queue at the position indicated by the *msg_prio* argument. A message with a
 26113 larger numeric value of *msg_prio* shall be inserted before messages with lower values of
 26114 *msg_prio*. A message shall be inserted after other messages in the queue, if any, with equal
 26115 *msg_prio*. The value of *msg_prio* shall be less than {MQ_PRIO_MAX}.

26116 If the specified message queue is full and O_NONBLOCK is not set in the message queue
 26117 description associated with *mqdes*, *mq_send()* shall block until space becomes available to
 26118 enqueue the message, or until *mq_send()* is interrupted by a signal. If more than one thread is
 26119 waiting to send when space becomes available in the message queue and the Priority Scheduling
 26120 option is supported, then the thread of the highest priority that has been waiting the longest
 26121 shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is
 26122 unblocked. If the specified message queue is full and O_NONBLOCK is set in the message
 26123 queue description associated with *mqdes*, the message shall not be queued and *mq_send()* shall
 26124 return an error.

26125 TMO The *mq_timedsend()* function shall add a message to the message queue specified by *mqdes* in the
 26126 manner defined for the *mq_send()* function. However, if the specified message queue is full and
 26127 O_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for
 26128 sufficient room in the queue shall be terminated when the specified timeout expires. If
 26129 O_NONBLOCK is set in the message queue description, this function shall be equivalent to
 26130 *mq_send()*.

26131 The timeout shall expire when the absolute time specified by *abs_timeout* passes, as measured by
 26132 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 26133 *abs_timeout*), or if the absolute time specified by *abs_timeout* has already been passed at the time
 26134 of the call.

26135 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock; if
 26136 the Timers option is not supported, the timeout shall be based on the system clock as returned
 26137 by the *time()* function.

26138 TMO The resolution of the timeout shall be the resolution of the clock on which it is based. The
 26139 *timespec* argument is defined in the <time.h> header.

26140 Under no circumstance shall the operation fail with a timeout if there is sufficient room in the
 26141 queue to add the message immediately. The validity of the *abs_timeout* parameter need not be
 26142 checked when there is sufficient room in the queue.

26143 RETURN VALUE

26144 TMO Upon successful completion, the *mq_send()* and *mq_timedsend()* functions shall return a value of
 26145 zero. Otherwise, no message shall be enqueued, the functions shall return `-1`, and *errno* shall be
 26146 set to indicate the error.

26147 ERRORS

26148 TMO The *mq_send()* and *mq_timedsend()* functions shall fail if:

26149 [EAGAIN] The `O_NONBLOCK` flag is set in the message queue description associated
 26150 with *mqdes*, and the specified message queue is full.

26151 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for writing.

26152 TMO [EINTR] A signal interrupted the call to *mq_send()* or *mq_timedsend()*.

26153 [EINVAL] The value of *msg_prio* was outside the valid range.

26154 TMO [EINVAL] The process or thread would have blocked, and the *abs_timeout* parameter
 26155 specified a nanoseconds field value less than zero or greater than or equal to
 26156 1 000 million.

26157 [EMSGSIZE] The specified message length, *msg_len*, exceeds the message size attribute of
 26158 the message queue.

26159 TMO [ETIMEDOUT] The `O_NONBLOCK` flag was not set when the message queue was opened,
 26160 but the timeout expired before the message could be added to the queue.

26161 EXAMPLES

26162 None.

26163 APPLICATION USAGE

26164 The value of the symbol `{MQ_PRIO_MAX}` limits the number of priority levels supported by the
 26165 application. Message priorities range from 0 to `{MQ_PRIO_MAX}-1`.

26166 RATIONALE

26167 None.

26168 FUTURE DIRECTIONS

26169 None.

26170 SEE ALSO

26171 *mq_open()*, *mq_receive()*, *mq_setattr()*, *mq_timedreceive()*, *time()*, the Base Definitions volume of
 26172 IEEE Std 1003.1-2001, `<mqqueue.h>`, `<time.h>`

26173 CHANGE HISTORY

26174 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26175 Issue 6

26176 The *mq_send()* function is marked as part of the Message Passing option.

26177 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an
 26178 implementation does not support the Message Passing option.

26179 The *mq_timedsend()* function is added for alignment with IEEE Std 1003.1d-1999.

26180 **NAME**26181 mq_setattr — set message queue attributes (**REALTIME**)26182 **SYNOPSIS**

26183 MSG #include <mqueue.h>

```
26184 int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
26185               struct mq_attr *restrict omqstat);
```

26186

26187 **DESCRIPTION**

26188 The *mq_setattr()* function shall set attributes associated with the open message queue
 26189 description referenced by the message queue descriptor specified by *mqdes*.

26190 The message queue attributes corresponding to the following members defined in the **mq_attr**
 26191 structure shall be set to the specified values upon successful completion of *mq_setattr()*:

26192 *mq_flags* The value of this member is the bitwise-logical OR of zero or more of
 26193 O_NONBLOCK and any implementation-defined flags.

26194 The values of the *mq_maxmsg*, *mq_msgsize*, and *mq_curmsgs* members of the **mq_attr** structure
 26195 shall be ignored by *mq_setattr()*.

26196 If *omqstat* is non-NULL, the *mq_setattr()* function shall store, in the location referenced by
 26197 *omqstat*, the previous message queue attributes and the current queue status. These values shall
 26198 be the same as would be returned by a call to *mq_getattr()* at that point.

26199 **RETURN VALUE**

26200 Upon successful completion, the function shall return a value of zero and the attributes of the
 26201 message queue shall have been changed as specified.

26202 Otherwise, the message queue attributes shall be unchanged, and the function shall return a
 26203 value of -1 and set *errno* to indicate the error.

26204 **ERRORS**

26205 The *mq_setattr()* function shall fail if:

26206 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

26207 **EXAMPLES**

26208 None.

26209 **APPLICATION USAGE**

26210 None.

26211 **RATIONALE**

26212 None.

26213 **FUTURE DIRECTIONS**

26214 None.

26215 **SEE ALSO**

26216 *mq_open()*, *mq_send()*, *mq_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base
 26217 Definitions volume of IEEE Std 1003.1-2001, <**mqueue.h**>

26218 **CHANGE HISTORY**

26219 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26220 **Issue 6**

- 26221 The *mq_setattr()* function is marked as part of the Message Passing option.
- 26222 The [ENOSYS] error condition has been removed as stubs need not be provided if an
26223 implementation does not support the Message Passing option.
- 26224 The *mq_timedsend()* function is added to the SEE ALSO section for alignment with
26225 IEEE Std 1003.1d-1999.
- 26226 The **restrict** keyword is added to the *mq_setattr()* prototype for alignment with the
26227 ISO / IEC 9899:1999 standard.

26228 **NAME**

26229 mq_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)

26230 **SYNOPSIS**

26231 MSG TMO #include <mqueue.h>

26232 #include <time.h>

```
26233       ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,  
26234                               size_t msg_len, unsigned *restrict msg_prio,  
26235                               const struct timespec *restrict abs_timeout);
```

26236

26237 **DESCRIPTION**

26238 Refer to *mq_receive()*.

26239 **NAME**26240 mq_timedsend — send a message to a message queue (**ADVANCED REALTIME**)26241 **SYNOPSIS**26242 MSG TMO `#include <mqueue.h>`26243 `#include <time.h>`

```
26244       int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,  
26245                       unsigned msg_prio, const struct timespec *abs_timeout);
```

26246

26247 **DESCRIPTION**26248 Refer to *mq_send()*.

26249 **NAME**26250 mq_unlink — remove a message queue (**REALTIME**)26251 **SYNOPSIS**

26252 MSG #include <mqueue.h>

26253 int mq_unlink(const char *name);

26254

26255 **DESCRIPTION**

26256 The *mq_unlink()* function shall remove the message queue named by the pathname *name*. After
 26257 a successful call to *mq_unlink()* with *name*, a call to *mq_open()* with *name* shall fail if the flag
 26258 O_CREAT is not set in *flags*. If one or more processes have the message queue open when
 26259 *mq_unlink()* is called, destruction of the message queue shall be postponed until all references to
 26260 the message queue have been closed.

26261 Calls to *mq_open()* to recreate the message queue may fail until the message queue is actually
 26262 removed. However, the *mq_unlink()* call need not block until all references have been closed; it
 26263 may return immediately.

26264 **RETURN VALUE**

26265 Upon successful completion, the function shall return a value of zero. Otherwise, the named
 26266 message queue shall be unchanged by this function call, and the function shall return a value of
 26267 -1 and set *errno* to indicate the error.

26268 **ERRORS**26269 The *mq_unlink()* function shall fail if:

26270 [EACCES] Permission is denied to unlink the named message queue.

26271 [ENAMETOOLONG]

26272 The length of the *name* argument exceeds {PATH_MAX} or a pathname
 26273 component is longer than {NAME_MAX}.

26274 [ENOENT] The named message queue does not exist.

26275 **EXAMPLES**

26276 None.

26277 **APPLICATION USAGE**

26278 None.

26279 **RATIONALE**

26280 None.

26281 **FUTURE DIRECTIONS**

26282 None.

26283 **SEE ALSO**

26284 *mq_close()*, *mq_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of
 26285 IEEE Std 1003.1-2001, <mqueue.h>

26286 **CHANGE HISTORY**

26287 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26288 **Issue 6**26289 The *mq_unlink()* function is marked as part of the Message Passing option.

26290 The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion,
 26291 the named message queue is unchanged by this function.

26292 The [ENOSYS] error condition has been removed as stubs need not be provided if an
26293 implementation does not support the Message Passing option.

26294 **NAME**

26295 mrnd48 — generate uniformly distributed pseudo-random signed long integers

26296 **SYNOPSIS**

26297 xSI #include <stdlib.h>

26298 long mrnd48(void);

26299

26300 **DESCRIPTION**

26301 Refer to *drand48()*.

26302 **NAME**

26303 msgctl — XSI message control operations

26304 **SYNOPSIS**26305 xsi `#include <sys/msg.h>`26306 `int msgctl(int msqid, int cmd, struct msqid_ds *buf);`

26307

26308 **DESCRIPTION**

26309 The *msgctl()* function operates on XSI message queues (see the Base Definitions volume of
 26310 IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function
 26311 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on
 26312 page 41).

26313 The *msgctl()* function shall provide message control operations as specified by *cmd*. The
 26314 following values for *cmd*, and the message control operations they specify, are:

26315 **IPC_STAT** Place the current value of each member of the **msqid_ds** data structure
 26316 associated with *msqid* into the structure pointed to by *buf*. The contents of this
 26317 structure are defined in **<sys/msg.h>**.

26318 **IPC_SET** Set the value of the following members of the **msqid_ds** data structure
 26319 associated with *msqid* to the corresponding value found in the structure
 26320 pointed to by *buf*:

26321 msg_perm.uid
 26322 msg_perm.gid
 26323 msg_perm.mode
 26324 msg_qbytes

26325 **IPC_SET** can only be executed by a process with appropriate privileges or that
 26326 has an effective user ID equal to the value of **msg_perm.cuid** or
 26327 **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*. Only a
 26328 process with appropriate privileges can raise the value of **msg_qbytes**.

26329 **IPC_RMID** Remove the message queue identifier specified by *msqid* from the system and
 26330 destroy the message queue and **msqid_ds** data structure associated with it.
 26331 **IPC_RMD** can only be executed by a process with appropriate privileges or
 26332 one that has an effective user ID equal to the value of **msg_perm.cuid** or
 26333 **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*.

26334 **RETURN VALUE**

26335 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return -1 and set *errno* to
 26336 indicate the error.

26337 **ERRORS**26338 The *msgctl()* function shall fail if:

26339 **[EACCES]** The argument *cmd* is **IPC_STAT** and the calling process does not have read
 26340 permission; see Section 2.7 (on page 39).

26341 **[EINVAL]** The value of *msqid* is not a valid message queue identifier; or the value of *cmd*
 26342 is not a valid command.

26343 **[EPERM]** The argument *cmd* is **IPC_RMID** or **IPC_SET** and the effective user ID of the
 26344 calling process is not equal to that of a process with appropriate privileges
 26345 and it is not equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the data
 26346 structure associated with *msqid*.

26347 [EPERM] The argument *cmd* is IPC_SET, an attempt is being made to increase to the
26348 value of **msg_qbytes**, and the effective user ID of the calling process does not
26349 have appropriate privileges.

26350 EXAMPLES

26351 None.

26352 APPLICATION USAGE

26353 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
26354 (IPC). Application developers who need to use IPC should design their applications so that
26355 modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to
26356 use the alternative interfaces.

26357 RATIONALE

26358 None.

26359 FUTURE DIRECTIONS

26360 None.

26361 SEE ALSO

26362 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq_close()*, *mq_getattr()*, *mq_notify()*,
26363 *mq_open()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_unlink()*, *msgget()*, *msgrcv()*, *msgsnd()*, the
26364 Base Definitions volume of IEEE Std 1003.1-2001, <sys/msg.h>

26365 CHANGE HISTORY

26366 First released in Issue 2. Derived from Issue 2 of the SVID.

26367 Issue 5

26368 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
26369 DIRECTIONS to a new APPLICATION USAGE section.

26370 **NAME**

26371 msgget — get the XSI message queue identifier

26372 **SYNOPSIS**

26373 xsi #include <sys/msg.h>

26374 int msgget(key_t key, int msgflg);

26375

26376 **DESCRIPTION**

26377 The *msgget()* function operates on XSI message queues (see the Base Definitions volume of
 26378 IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function
 26379 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on
 26380 page 41).

26381 The *msgget()* function shall return the message queue identifier associated with the argument
 26382 *key*.

26383 A message queue identifier, associated message queue, and data structure (see <sys/msg.h>),
 26384 shall be created for the argument *key* if one of the following is true:

- 26385 • The argument *key* is equal to IPC_PRIVATE.
- 26386 • The argument *key* does not already have a message queue identifier associated with it, and
 26387 (*msgflg* & IPC_CREAT) is non-zero.

26388 Upon creation, the data structure associated with the new message queue identifier shall be
 26389 initialized as follows:

- 26390 • **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** shall be set equal to the
 26391 effective user ID and effective group ID, respectively, of the calling process.
- 26392 • The low-order 9 bits of **msg_perm.mode** shall be set equal to the low-order 9 bits of *msgflg*.
- 26393 • **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** shall be set equal to 0.
- 26394 • **msg_ctime** shall be set equal to the current time.
- 26395 • **msg_qbytes** shall be set equal to the system limit.

26396 **RETURN VALUE**

26397 Upon successful completion, *msgget()* shall return a non-negative integer, namely a message
 26398 queue identifier. Otherwise, it shall return -1 and set *errno* to indicate the error.

26399 **ERRORS**26400 The *msgget()* function shall fail if:

26401	[EACCES]	A message queue identifier exists for the argument <i>key</i> , but operation
26402		permission as specified by the low-order 9 bits of <i>msgflg</i> would not be granted;
26403		see Section 2.7 (on page 39).
26404	[EEXIST]	A message queue identifier exists for the argument <i>key</i> but ((<i>msgflg</i> &
26405		IPC_CREAT) && (<i>msgflg</i> & IPC_EXCL)) is non-zero.
26406	[ENOENT]	A message queue identifier does not exist for the argument <i>key</i> and (<i>msgflg</i> &
26407		IPC_CREAT) is 0.
26408	[ENOSPC]	A message queue identifier is to be created but the system-imposed limit on
26409		the maximum number of allowed message queue identifiers system-wide
26410		would be exceeded.

26411 **EXAMPLES**

26412 None.

26413 **APPLICATION USAGE**

26414 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
26415 (IPC). Application developers who need to use IPC should design their applications so that
26416 modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to
26417 use the alternative interfaces.

26418 **RATIONALE**

26419 None.

26420 **FUTURE DIRECTIONS**

26421 None.

26422 **SEE ALSO**

26423 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq_close()*, *mq_getattr()*, *mq_notify()*,
26424 *mq_open()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_unlink()*, *msgctl()*, *msgrcv()*, *msgsnd()*, the
26425 Base Definitions volume of IEEE Std 1003.1-2001, <sys/msg.h>

26426 **CHANGE HISTORY**

26427 First released in Issue 2. Derived from Issue 2 of the SVID.

26428 **Issue 5**

26429 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
26430 DIRECTIONS to a new APPLICATION USAGE section.

26431 **NAME**

26432 msgrcv — XSI message receive operation

26433 **SYNOPSIS**26434 XSI

```
#include <sys/msg.h>
```

```
26435 ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
26436               int msgflg);
26437
```

26438 **DESCRIPTION**

26439 The *msgrcv()* function operates on XSI message queues (see the Base Definitions volume of
 26440 IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function
 26441 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on
 26442 page 41).

26443 The *msgrcv()* function shall read a message from the queue associated with the message queue
 26444 identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

26445 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains
 26446 first a field of type **long** specifying the type of the message, and then a data portion that holds
 26447 the data bytes of the message. The structure below is an example of what this user-defined
 26448 buffer might look like:

```
26449 struct mymsg {
26450     long    mtype;      /* Message type. */
26451     char    mtext[1];   /* Message text. */
26452 }
```

26453 The structure member *mtype* is the received message's type as specified by the sending process.

26454 The structure member *mtext* is the text of the message.

26455 The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated
 26456 to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG_NOERROR) is non-zero. The
 26457 truncated part of the message shall be lost and no indication of the truncation shall be given to
 26458 the calling process.

26459 If the value of *msgsz* is greater than {SSIZE_MAX}, the result is implementation-defined.

26460 The argument *msgtyp* specifies the type of message requested as follows:

- 26461 • If *msgtyp* is 0, the first message on the queue shall be received.
- 26462 • If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received.
- 26463 • If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the
 26464 absolute value of *msgtyp* shall be received.

26465 The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the
 26466 queue. These are as follows:

- 26467 • If (*msgflg* & IPC_NOWAIT) is non-zero, the calling thread shall return immediately with a
 26468 return value of -1 and *errno* set to [ENOMSG].
- 26469 • If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the
 26470 following occurs:
 - 26471 — A message of the desired type is placed on the queue.
 - 26472 — The message queue identifier *msqid* is removed from the system; when this occurs, *errno*
 26473 shall be set equal to [EIDRM] and -1 shall be returned.

26474 — The calling thread receives a signal that is to be caught; in this case a message is not
 26475 received and the calling thread resumes execution in the manner prescribed in *sigaction()*.

26476 Upon successful completion, the following actions are taken with respect to the data structure
 26477 associated with *msqid*:

- 26478 • **msg_qnum** shall be decremented by 1.
- 26479 • **msg_lrpid** shall be set equal to the process ID of the calling process.
- 26480 • **msg_rtime** shall be set equal to the current time.

26481 RETURN VALUE

26482 Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually
 26483 placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return
 26484 (**ssize_t**)−1, and *errno* shall be set to indicate the error.

26485 ERRORS

26486 The *msgrcv()* function shall fail if:

26487	[E2BIG]	The value of <i>mtext</i> is greater than <i>msgsz</i> and (<i>msgflg</i> & MSG_NOERROR) is 0.
26488	[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 39).
26489		
26490	[EIDRM]	The message queue identifier <i>msqid</i> is removed from the system.
26491	[EINTR]	The <i>msgrcv()</i> function was interrupted by a signal.
26492	[EINVAL]	<i>msqid</i> is not a valid message queue identifier.
26493	[ENOMSG]	The queue does not contain a message of the desired type and (<i>msgflg</i> & IPC_NOWAIT) is non-zero.
26494		

26495 EXAMPLES

26496 Receiving a Message

26497 The following example receives the first message on the queue (based on the value of the *msgtyp*
 26498 argument, 0). The queue is identified by the *msqid* argument (assuming that the value has
 26499 previously been set). This call specifies that an error should be reported if no message is
 26500 available, but not if the message is too large. The message size is calculated directly using the
 26501 *sizeof* operator.

```

26502 #include <sys/msg.h>
26503 ...
26504 int result;
26505 int msqid;
26506 struct message {
26507     long type;
26508     char text[20];
26509 } msg;
26510 long msgtyp = 0;
26511 ...
26512 result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
26513               msgtyp, MSG_NOERROR | IPC_NOWAIT);
  
```


26514 APPLICATION USAGE

26515 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
26516 (IPC). Application developers who need to use IPC should design their applications so that
26517 modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to
26518 use the alternative interfaces.

26519 RATIONALE

26520 None.

26521 FUTURE DIRECTIONS

26522 None.

26523 SEE ALSO

26524 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq_close()*, *mq_getattr()*, *mq_notify()*,
26525 *mq_open()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_unlink()*, *msgctl()*, *msgget()*, *msgsnd()*,
26526 *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/msg.h>**

26527 CHANGE HISTORY

26528 First released in Issue 2. Derived from Issue 2 of the SVID.

26529 Issue 5

26530 The type of the return value is changed from **int** to **ssize_t**, and a warning is added to the
26531 DESCRIPTION about values of *msgsz* larger the {SSIZE_MAX}.

26532 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
26533 DIRECTIONS to the APPLICATION USAGE section.

26534 Issue 6

26535 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

26536 **NAME**

26537 msgsnd — XSI message send operation

26538 **SYNOPSIS**26539 xsi `#include <sys/msg.h>`26540 `int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);`

26541

26542 **DESCRIPTION**

26543 The `msgsnd()` function operates on XSI message queues (see the Base Definitions volume of
 26544 IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function
 26545 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on
 26546 page 41).

26547 The `msgsnd()` function shall send a message to the queue associated with the message queue
 26548 identifier specified by `msqid`.

26549 The application shall ensure that the argument `msgp` points to a user-defined buffer that contains
 26550 first a field of type **long** specifying the type of the message, and then a data portion that holds
 26551 the data bytes of the message. The structure below is an example of what this user-defined
 26552 buffer might look like:

```
26553 struct mymsg {
26554     long    mtype;           /* Message type. */
26555     char    mtext[1];       /* Message text. */
26556 }
```

26557 The structure member `mtype` is a non-zero positive type **long** that can be used by the receiving
 26558 process for message selection.

26559 The structure member `mtext` is any text of length `msgsz` bytes. The argument `msgsz` can range
 26560 from 0 to a system-imposed maximum.

26561 The argument `msgflg` specifies the action to be taken if one or more of the following is true:

- 26562 • The number of bytes already on the queue is equal to **msg_qbytes**; see `<sys/msg.h>`.
- 26563 • The total number of messages on all queues system-wide is equal to the system-imposed
 26564 limit.

26565 These actions are as follows:

- 26566 • If `(msgflg & IPC_NOWAIT)` is non-zero, the message shall not be sent and the calling thread
 26567 shall return immediately.
- 26568 • If `(msgflg & IPC_NOWAIT)` is 0, the calling thread shall suspend execution until one of the
 26569 following occurs:
 - 26570 — The condition responsible for the suspension no longer exists, in which case the message
 26571 is sent.
 - 26572 — The message queue identifier `msqid` is removed from the system; when this occurs, `errno`
 26573 shall be set equal to `[EIDRM]` and `-1` shall be returned.
 - 26574 — The calling thread receives a signal that is to be caught; in this case the message is not
 26575 sent and the calling thread resumes execution in the manner prescribed in `sigaction()`.

26576 Upon successful completion, the following actions are taken with respect to the data structure
 26577 associated with `msqid`; see `<sys/msg.h>`:

- 26578 • **msg_qnum** shall be incremented by 1.
- 26579 • **msg_lspid** shall be set equal to the process ID of the calling process.
- 26580 • **msg_stime** shall be set equal to the current time.

26581 RETURN VALUE

26582 Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent,
 26583 *msgsnd()* shall return -1, and *errno* shall be set to indicate the error.

26584 ERRORS

26585 The *msgsnd()* function shall fail if:

26586 26587	[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 39).
26588 26589	[EAGAIN]	The message cannot be sent for one of the reasons cited above and (<i>msgflg</i> & <i>IPC_NOWAIT</i>) is non-zero.
26590	[EIDRM]	The message queue identifier <i>msqid</i> is removed from the system.
26591	[EINTR]	The <i>msgsnd()</i> function was interrupted by a signal.
26592 26593 26594	[EINVAL]	The value of <i>msqid</i> is not a valid message queue identifier, or the value of <i>mtype</i> is less than 1; or the value of <i>msgsz</i> is less than 0 or greater than the system-imposed limit.

26595 EXAMPLES**26596 Sending a Message**

26597 The following example sends a message to the queue identified by the *msqid* argument
 26598 (assuming that value has previously been set). This call specifies that an error should be
 26599 reported if no message is available. The message size is calculated directly using the *sizeof*
 26600 operator.

```

26601 #include <sys/msg.h>
26602 ...
26603 int result;
26604 int msqid;
26605 struct message {
26606     long type;
26607     char text[20];
26608 } msg;

26609 msg.type = 1;
26610 strcpy(msg.text, "This is message 1");
26611 ...
26612 result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
  
```

26613 APPLICATION USAGE

26614 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
 26615 (IPC). Application developers who need to use IPC should design their applications so that
 26616 modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to
 26617 use the alternative interfaces.

26618 **RATIONALE**

26619 None.

26620 **FUTURE DIRECTIONS**

26621 None.

26622 **SEE ALSO**

26623 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq_close()*, *mq_getattr()*, *mq_notify()*,
26624 *mq_open()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*,
26625 *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/msg.h>**

26626 **CHANGE HISTORY**

26627 First released in Issue 2. Derived from Issue 2 of the SVID.

26628 **Issue 5**

26629 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
26630 DIRECTIONS to a new APPLICATION USAGE section.

26631 **Issue 6**

26632 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

26633 **NAME**

26634 msync — synchronize memory with physical storage

26635 **SYNOPSIS**

26636 MF SIO #include <sys/mman.h>

26637 int msync(void *addr, size_t len, int flags);

26638

26639 **DESCRIPTION**

26640 The *msync()* function shall write all modified data to permanent storage locations, if any, in
 26641 those whole pages containing any part of the address space of the process starting at address
 26642 *addr* and continuing for *len* bytes. If no such storage exists, *msync()* need not have any effect. If
 26643 requested, the *msync()* function shall then invalidate cached copies of data.

26644 The implementation shall require that *addr* be a multiple of the page size as returned by
 26645 *sysconf()*.

26646 For mappings to files, the *msync()* function shall ensure that all write operations are completed
 26647 as defined for synchronized I/O data integrity completion. It is unspecified whether the
 26648 implementation also writes out other file attributes. When the *msync()* function is called on
 26649 MAP_PRIVATE mappings, any modified data shall not be written to the underlying object and
 26650 shall not cause such data to be made visible to other processes. It is unspecified whether data in
 26651 SHM|TYM MAP_PRIVATE mappings has any permanent storage locations. The effect of *msync()* on a
 26652 shared memory object or a typed memory object is unspecified. The behavior of this function is
 26653 unspecified if the mapping was not established by a call to *mmap()*.

26654 The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following
 26655 flags defined in the <sys/mman.h> header:

26656

26657

26658

26659

26660

Symbolic Constant	Description
MS_ASYNC	Perform asynchronous writes.
MS_SYNC	Perform synchronous writes.
MS_INVALIDATE	Invalidate cached data.

26661 When MS_ASYNC is specified, *msync()* shall return immediately once all the write operations
 26662 are initiated or queued for servicing; when MS_SYNC is specified, *msync()* shall not return until
 26663 all write operations are completed as defined for synchronized I/O data integrity completion.
 26664 Either MS_ASYNC or MS_SYNC is specified, but not both.

26665 When MS_INVALIDATE is specified, *msync()* shall invalidate all cached copies of mapped data
 26666 that are inconsistent with the permanent storage locations such that subsequent references shall
 26667 obtain data that was consistent with the permanent storage locations sometime between the call
 26668 to *msync()* and the first subsequent memory reference to the data.

26669 If *msync()* causes any write to a file, the file's *st_ctime* and *st_mtime* fields shall be marked for
 26670 update.

26671 **RETURN VALUE**

26672 Upon successful completion, *msync()* shall return 0; otherwise, it shall return *-1* and set *errno* to
 26673 indicate the error.

26674 **ERRORS**26675 The *msync()* function shall fail if:

26676 [EBUSY] Some or all of the addresses in the range starting at *addr* and continuing for *len*
 26677 bytes are locked, and MS_INVALIDATE is specified.

26678 [EINVAL] The value of *flags* is invalid.

26679 [EINVAL] The value of *addr* is not a multiple of the page size {PAGESIZE}.

26680 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process or specify one or more pages that are not mapped.

26681

26682

26683 EXAMPLES

26684 None.

26685 APPLICATION USAGE

26686 The *msync()* function is only supported if the Memory Mapped Files option and the Synchronized Input and Output option are supported, and thus need not be available on all implementations.

26687

26688

26689 The *msync()* function should be used by programs that require a memory object to be in a known state; for example, in building transaction facilities.

26690

26691 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that *msync()* is the only control over when pages are or are not written to disk.

26692

26693 RATIONALE

26694 The *msync()* function writes out data in a mapped region to the permanent storage for the underlying object. The call to *msync()* ensures data integrity of the file.

26695

26696 After the data is written out, any cached data may be invalidated if the MS_INVALIDATE flag was specified. This is useful on systems that do not support read/write consistency.

26697

26698 FUTURE DIRECTIONS

26699 None.

26700 SEE ALSO

26701 *mmap()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/mman.h>

26702 CHANGE HISTORY

26703 First released in Issue 4, Version 2.

26704 Issue 5

26705 Moved from X/OPEN UNIX extension to BASE.

26706 Aligned with *msync()* in the POSIX Realtime Extension as follows:

- 26707 • The DESCRIPTION is extensively reworded.
- 26708 • [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.

26709 Issue 6

26710 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input and Output options.

26711

26712 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 26713 • The [EBUSY] mandatory error condition is added.

26714 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

26715

- 26716 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
- 26717
- 26718 • The second [EINVAL] error condition is made mandatory.

26719 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to
26720 typed memory objects.

26721 NAME

26722 munlock — unlock a range of process address space

26723 SYNOPSIS

26724 MLR #include <sys/mman.h>

26725 int munlock(const void *addr, size_t len);

26726

26727 DESCRIPTION

26728 Refer to *mlock()*.

26729 **NAME**

26730 munlockall — unlock the address space of a process

26731 **SYNOPSIS**

26732 ML #include <sys/mman.h>

26733 int munlockall(void);

26734

26735 **DESCRIPTION**26736 Refer to *mlockall()*.

26737 **NAME**

26738 munmap — unmap pages of memory

26739 **SYNOPSIS**

26740 MC3 #include <sys/mman.h>

26741 int munmap(void *addr, size_t len);

26742

26743 **DESCRIPTION**

26744 The *munmap()* function shall remove any mappings for those entire pages containing any part of
 26745 the address space of the process starting at *addr* and continuing for *len* bytes. Further references
 26746 to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no
 26747 mappings in the specified address range, then *munmap()* has no effect.

26748 The implementation shall require that *addr* be a multiple of the page size {PAGESIZE}.

26749 If a mapping to be removed was private, any modifications made in this address range shall be
 26750 discarded.

26751 ML|MLR Any memory locks (see *mlock()* and *mlockall()*) associated with this address range shall be
 26752 removed, as if by an appropriate call to *munlock()*.

26753 TYM If a mapping removed from a typed memory object causes the corresponding address range of
 26754 the memory pool to be inaccessible by any process in the system except through allocatable
 26755 mappings (that is, mappings of typed memory objects opened with the
 26756 POSIX_TYPED_MEM_MAP_ALLOCATABLE flag), then that range of the memory pool shall
 26757 become deallocated and may become available to satisfy future typed memory allocation
 26758 requests.

26759 A mapping removed from a typed memory object opened with the
 26760 POSIX_TYPED_MEM_MAP_ALLOCATABLE flag shall not affect in any way the availability of
 26761 that typed memory for allocation.

26762 The behavior of this function is unspecified if the mapping was not established by a call to
 26763 *mmap()*.

26764 **RETURN VALUE**

26765 Upon successful completion, *munmap()* shall return 0; otherwise, it shall return -1 and set *errno*
 26766 to indicate the error.

26767 **ERRORS**26768 The *munmap()* function shall fail if:

26769 [EINVAL] Addresses in the range [*addr*,*addr*+*len*) are outside the valid range for the
 26770 address space of a process.

26771 [EINVAL] The *len* argument is 0.

26772 [EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

26773 **EXAMPLES**

26774 None.

26775 **APPLICATION USAGE**

26776 The *munmap()* function is only supported if the Memory Mapped Files option or the Shared
 26777 Memory Objects option is supported.

26778 **RATIONALE**26779 The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

26780 It is possible that an application has applied process memory locking to a region that contains
 26781 shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary,
 26782 causes those locks to be removed.

26783 **FUTURE DIRECTIONS**

26784 None.

26785 **SEE ALSO**

26786 *mlock()*, *mlockall()*, *mmap()*, *posix_typed_mem_open()*, *sysconf()*, the Base Definitions volume of
 26787 IEEE Std 1003.1-2001, <signal.h>, <sys/mman.h>

26788 **CHANGE HISTORY**

26789 First released in Issue 4, Version 2.

26790 **Issue 5**

26791 Moved from X/OPEN UNIX extension to BASE.

26792 Aligned with *munmap()* in the POSIX Realtime Extension as follows:

- 26793 • The DESCRIPTION is extensively reworded.
- 26794 • The SIGBUS error is no longer permitted to be generated.

26795 **Issue 6**

26796 The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory
 26797 Objects option.

26798 The following new requirements on POSIX implementations derive from alignment with the
 26799 Single UNIX Specification:

- 26800 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of
 26801 the page size.
- 26802 • The [EINVAL] error conditions are added.

26803 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 26804 • Semantics for typed memory objects are added to the DESCRIPTION.
- 26805 • The *posix_typed_mem_open()* function is added to the SEE ALSO section.

26806 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code |
 26807 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM). |

26808 **NAME**

26809 nan, nanf, nanl — return quiet NaN

26810 **SYNOPSIS**

26811 #include <math.h>

26812 double nan(const char *tagp);

26813 float nanf(const char *tagp);

26814 long double nanl(const char *tagp);

26815 **DESCRIPTION**

26816 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 26817 conflict between the requirements described here and the ISO C standard is unintentional. This
 26818 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

26819 The function call *nan*("n-char-sequence") shall be equivalent to:

26820 strtod("NAN(n-char-sequence)", (char **) NULL);

26821 The function call *nan*("") shall be equivalent to:

26822 strtod("NAN()", (char **) NULL)

26823 If *tagp* does not point to an *n-char* sequence or an empty string, the function call shall be
 26824 equivalent to:

26825 strtod("NAN", (char **) NULL)

26826 Function calls to *nanf*() and *nanl*() are equivalent to the corresponding function calls to *strtof*()
 26827 and *strtold*().

26828 **RETURN VALUE**26829 These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

26830 If the implementation does not support quiet NaNs, these functions shall return zero.

26831 **ERRORS**

26832 No errors are defined.

26833 **EXAMPLES**

26834 None.

26835 **APPLICATION USAGE**

26836 None.

26837 **RATIONALE**

26838 None.

26839 **FUTURE DIRECTIONS**

26840 None.

26841 **SEE ALSO**26842 *strtod*(), *strtold*(), the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>26843 **CHANGE HISTORY**

26844 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26845 **NAME**26846 nanosleep — high resolution sleep (**REALTIME**)26847 **SYNOPSIS**

26848 TMR #include <time.h>

26849 int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);

26850

26851 **DESCRIPTION**

26852 The *nanosleep()* function shall cause the current thread to be suspended from execution until
 26853 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the
 26854 calling thread, and its action is to invoke a signal-catching function or to terminate the process.
 26855 The suspension time may be longer than requested because the argument value is rounded up to
 26856 an integer multiple of the sleep resolution or because of the scheduling of other activity by the
 26857 system. But, except for the case of being interrupted by a signal, the suspension time shall not be
 26858 less than the time specified by *rqtp*, as measured by the system clock `CLOCK_REALTIME`.

26859 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.26860 **RETURN VALUE**26861 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall
26862 be zero.

26863 If the *nanosleep()* function returns because it has been interrupted by a signal, it shall return a
 26864 value of `-1` and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the
 26865 **timespec** structure referenced by it is updated to contain the amount of time remaining in the
 26866 interval (the requested time minus the time actually slept). If the *rmtp* argument is NULL, the
 26867 remaining time is not returned.

26868 If *nanosleep()* fails, it shall return a value of `-1` and set *errno* to indicate the error.26869 **ERRORS**26870 The *nanosleep()* function shall fail if:26871 [EINTR] The *nanosleep()* function was interrupted by a signal.

26872 [EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than
 26873 or equal to 1 000 million.

26874 **EXAMPLES**

26875 None.

26876 **APPLICATION USAGE**

26877 None.

26878 **RATIONALE**

26879 It is common to suspend execution of a process for an interval in order to poll the status of a
 26880 non-interrupting function. A large number of actual needs can be met with a simple extension to
 26881 *sleep()* that provides finer resolution.

26882 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the
 26883 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,
 26884 it is possible to write such a routine using no static storage and reserving no system facilities.
 26885 Although it is possible to write a function with similar functionality to *sleep()* using the
 26886 remainder of the *timer_**() functions, such a function requires the use of signals and the
 26887 reservation of some signal number. This volume of IEEE Std 1003.1-2001 requires that
 26888 *nanosleep()* be non-intrusive of the signals function.

26889 The *nanosleep()* function shall return a value of 0 on success and –1 on failure or if interrupted.
26890 This latter case is different from *sleep()*. This was done because the remaining time is returned
26891 via an argument structure pointer, *rmtp*, instead of as the return value.

26892 **FUTURE DIRECTIONS**

26893 None.

26894 **SEE ALSO**

26895 *clock_nanosleep()*, *sleep()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**> |

26896 **CHANGE HISTORY**

26897 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26898 **Issue 6**

26899 The *nanosleep()* function is marked as part of the Timers option.

26900 The [ENOSYS] error condition has been removed as stubs need not be provided if an
26901 implementation does not support the Timers option.

26902 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO |
26903 section to include the *clock_nanosleep()* function. |

26904 **NAME**

26905 nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

26906 **SYNOPSIS**

26907 #include <math.h>

26908 double nearbyint(double x);

26909 float nearbyintf(float x);

26910 long double nearbyintl(long double x);

26911 **DESCRIPTION**

26912 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26913 conflict between the requirements described here and the ISO C standard is unintentional. This
 26914 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

26915 These functions shall round their argument to an integer value in floating-point format, using
 26916 the current rounding direction and without raising the inexact floating-point exception.

26917 An application wishing to check for error situations should set *errno* to zero and call
 26918 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 26919 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 26920 zero, an error has occurred.

26921 **RETURN VALUE**

26922 Upon successful completion, these functions shall return the rounded integer value.

26923 MX If *x* is NaN, a NaN shall be returned.26924 If *x* is ± 0 , ± 0 shall be returned.26925 If *x* is $\pm \text{Inf}$, *x* shall be returned.

26926 XSI If the correct value would cause overflow, a range error shall occur and *nearbyint*(), *nearbyintf*(),
 26927 and *nearbyintl*() shall return the value of the macro $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and
 26928 $\pm \text{HUGE\_VALL}$ (with the same sign as *x*), respectively.

26929 **ERRORS**

26930 These functions shall fail if:

26931 XSI **Range Error** The result would cause an overflow.

26932 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 26933 then *errno* shall be set to [ERANGE]. If the integer expression
 26934 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 26935 floating-point exception shall be raised.

26936 **EXAMPLES**

26937 None.

26938 **APPLICATION USAGE**

26939 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 26940 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

26941 **RATIONALE**

26942 None.

26943 **FUTURE DIRECTIONS**

26944 None.

26945 **SEE ALSO**

26946 *feclearexcept()*, *fetetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,
26947 Treatment of Error Conditions for Mathematical Functions, **<math.h>**

26948 **CHANGE HISTORY**

26949 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26950 **NAME**

26951 nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable
 26952 floating-point number

26953 **SYNOPSIS**

26954 #include <math.h>

26955 double nextafter(double x, double y);
 26956 float nextafterf(float x, float y);
 26957 long double nextafterl(long double x, long double y);
 26958 double nexttoward(double x, long double y);
 26959 float nexttowardf(float x, long double y);
 26960 long double nexttowardl(long double x, long double y);

26961 **DESCRIPTION**

26962 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 26963 conflict between the requirements described here and the ISO C standard is unintentional. This
 26964 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

26965 The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall compute the next representable
 26966 floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter()* shall
 26967 return the largest representable floating-point number less than *x*. The *nextafter()*, *nextafterf()*,
 26968 and *nextafterl()* functions shall return *y* if *x* equals *y*.

26969 The *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions shall be equivalent to the
 26970 corresponding *nextafter()* functions, except that the second parameter shall have type **long**
 26971 **double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

26972 An application wishing to check for error situations should set *errno* to zero and call
 26973 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 26974 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 26975 zero, an error has occurred.

26976 **RETURN VALUE**

26977 Upon successful completion, these functions shall return the next representable floating-point
 26978 value following *x* in the direction of *y*.

26979 If *x*==*y*, *y* (of the type *x*) shall be returned.

26980 If *x* is finite and the correct function value would overflow, a range error shall occur and
 26981 $\pm$ HUGE_VAL, $\pm$ HUGE_VALF, and $\pm$ HUGE_VALL (with the same sign as *x*) shall be returned as
 26982 appropriate for the return type of the function.

26983 **MX** If *x* or *y* is NaN, a NaN shall be returned.

26984 If *x*!=*y* and the correct function value is subnormal, zero, or underflows, a range error shall
 26985 occur, and either the correct function value (if representable) or 0.0 shall be returned.

26986 **ERRORS**

26987 These functions shall fail if:

26988 **Range Error** The correct value overflows.

26989 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 26990 then *errno* shall be set to [ERANGE]. If the integer expression
 26991 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 26992 floating-point exception shall be raised.

26993 **MX** **Range Error** The correct value is subnormal or underflows.

26994 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,
 26995 then *errno* shall be set to [ERANGE]. If the integer expression
 26996 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the underflow
 26997 floating-point exception shall be raised.

26998 **EXAMPLES**

26999 None.

27000 **APPLICATION USAGE**

27001 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
 27002 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

27003 **RATIONALE**

27004 None.

27005 **FUTURE DIRECTIONS**

27006 None.

27007 **SEE ALSO**

27008 *feclearexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,
 27009 Treatment of Error Conditions for Mathematical Functions, **<math.h>**

27010 **CHANGE HISTORY**

27011 First released in Issue 4, Version 2.

27012 **Issue 5**

27013 Moved from X/OPEN UNIX extension to BASE.

27014 **Issue 6**

27015 The *nextafter()* function is no longer marked as an extension.

27016 The *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions are added
 27017 for alignment with the ISO/IEC 9899:1999 standard.

27018 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 27019 revised to align with the ISO/IEC 9899:1999 standard.

27020 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 27021 marked.

27022 **NAME**

27023 nftw — walk a file tree

27024 **SYNOPSIS**

27025 xSI #include <ftw.h>

```
27026 int nftw(const char *path, int (*fn)(const char *,
27027     const struct stat *, int, struct FTW *), int depth, int flags);
27028
```

27029 **DESCRIPTION**

27030 The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The *nftw()*
 27031 function has a similar effect to *ftw()* except that it takes an additional argument *flags*, which is a
 27032 bitwise-inclusive OR of zero or more of the following flags:

27033 **FTW_CHDIR** If set, *nftw()* shall change the current working directory to each directory as it
 27034 reports files in that directory. If clear, *nftw()* shall not change the current
 27035 working directory.

27036 **FTW_DEPTH** If set, *nftw()* shall report all files in a directory before reporting the directory
 27037 itself. If clear, *nftw()* shall report any directory before reporting the files in that
 27038 directory.

27039 **FTW_MOUNT** If set, *nftw()* shall only report files in the same file system as *path*. If clear,
 27040 *nftw()* shall report all files encountered during the walk.

27041 **FTW_PHYS** If set, *nftw()* shall perform a physical walk and shall not follow symbolic links.

27042 If **FTW_PHYS** is clear and **FTW_DEPTH** is set, *nftw()* shall follow links instead of reporting
 27043 them, but shall not report any directory that would be a descendant of itself. If **FTW_PHYS** is
 27044 clear and **FTW_DEPTH** is clear, *nftw()* shall follow links instead of reporting them, but shall not
 27045 report the contents of any directory that would be a descendant of itself.

27046 At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

- 27047 • The first argument is the pathname of the object.
- 27048 • The second argument is a pointer to the **stat** buffer containing information on the object.
- 27049 • The third argument is an integer giving additional information. Its value is one of the
 27050 following:

27051 **FTW_F** The object is a file.

27052 **FTW_D** The object is a directory.

27053 **FTW_DP** The object is a directory and subdirectories have been visited. (This condition
 27054 shall only occur if the **FTW_DEPTH** flag is included in *flags*.)

27055 **FTW_SL** The object is a symbolic link. (This condition shall only occur if the **FTW_PHYS**
 27056 flag is included in *flags*.)

27057 **FTW_SLN** The object is a symbolic link that does not name an existing file. (This
 27058 condition shall only occur if the **FTW_PHYS** flag is not included in *flags*.)

27059 **FTW_DNR** The object is a directory that cannot be read. The *fn* function shall not be called
 27060 for any of its descendants.

27061 **FTW_NS** The *stat()* function failed on the object because of lack of appropriate
 27062 permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat()* for any
 27063 other reason is considered an error and *nftw()* shall return -1 .

27064 • The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the
 27065 object's filename in the pathname passed as the first argument to *fn*. The value of **level**
 27066 indicates depth relative to the root of the walk, where the root level is 0.

27067 The results are unspecified if the application-supplied *fn* function does not preserve the current
 27068 working directory.

27069 The argument *depth* sets the maximum number of file descriptors that shall be used by *nftw()*
 27070 while traversing the file tree. At most one file descriptor shall be used for each directory level.

27071 The *nftw()* function need not be reentrant. A function that is not required to be reentrant is not
 27072 required to be thread-safe.

27073 RETURN VALUE

27074 The *nftw()* function shall continue until the first of the following conditions occurs:

- 27075 • An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that value.
- 27076 • The *nftw()* function detects an error other than [EACCES] (see FTW_DNR and FTW_NS
 27077 above), in which case *nftw()* shall return -1 and set *errno* to indicate the error.
- 27078 • The tree is exhausted, in which case *nftw()* shall return 0.

27079 ERRORS

27080 The *nftw()* function shall fail if:

27081 [EACCES] Search permission is denied for any component of *path* or read permission is
 27082 denied for *path*, or *fn* returns -1 and does not reset *errno*.

27083 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 27084 argument.

27085 [ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname
 27086 component is longer than {NAME_MAX}.
 27087

27088 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

27089 [ENOTDIR] A component of *path* is not a directory.

27090 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current
 27091 programming environment for one or more files found in the file hierarchy.

27092 The *nftw()* function may fail if:

27093 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 27094 resolution of the *path* argument.

27095 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

27096 [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result
 27097 whose length exceeds {PATH_MAX}.
 27098

27099 [ENFILE] Too many files are currently open in the system.

27100 In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

27101 **EXAMPLES**

27102 The following example walks the **/tmp** directory and its subdirectories, calling the *nftw()*
 27103 function for every directory entry, to a maximum of 5 levels deep.

```
27104       #include <ftw.h>
27105       ...
27106       int nftwfunc(const char *, const struct stat *, int, struct FTW *);
27107       int nftwfunc(const char *filename, const struct stat *statptr,
27108                    int fileflags, struct FTW *pftw)
27109       {
27110           return 0;
27111       }
27112       ...
27113       char *startpath = "/tmp";
27114       int depth = 5;
27115       int flags = FTW_CHDIR | FTW_DEPTH | FTW_MOUNT;
27116       int ret;
27117       ret = nftw(startpath, nftwfunc, depth, flags);
```

27118 **APPLICATION USAGE**

27119 None.

27120 **RATIONALE**

27121 None.

27122 **FUTURE DIRECTIONS**

27123 None.

27124 **SEE ALSO**

27125 *lstat()*, *opendir()*, *readdir()*, *stat()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<ftw.h>**

27126 **CHANGE HISTORY**

27127 First released in Issue 4, Version 2.

27128 **Issue 5**

27129 Moved from X/OPEN UNIX extension to BASE.

27130 In the DESCRIPTION, the definition of the *depth* argument is clarified.

27131 **Issue 6**

27132 The Open Group Base Resolution bwg97-003 is applied.

27133 The ERRORS section is updated as follows:

- 27134 • The wording of the mandatory [ELOOP] error condition is updated.
- 27135 • A second optional [ELOOP] error condition is added.
- 27136 • The [Eoverflow] mandatory error condition is added.

27137 Text is added to the DESCRIPTION to say that the *nftw()* function need not be reentrant and
 27138 that the results are unspecified if the application-supplied *fn* function does not preserve the
 27139 current working directory.

27140 **NAME**

27141 *nice* — change the nice value of a process

27142 **SYNOPSIS**

27143 XSI `#include <unistd.h>`

27144 `int nice(int incr);`

27145

27146 **DESCRIPTION**

27147 The *nice()* function shall add the value of *incr* to the nice value of the calling process. A process' nice value is a non-negative number for which a more positive value shall result in less favorable scheduling.

27150 A maximum nice value of 2*{NZERO}-1 and a minimum nice value of 0 shall be imposed by the system. Requests for values above or below these limits shall result in the nice value being set to the corresponding limit. Only a process with appropriate privileges can lower the nice value.

27153 PS|TPS Calling the *nice()* function has no effect on the priority of processes or threads with policy SCHED_FIFO or SCHED_RR. The effect on processes or threads with other scheduling policies is implementation-defined.

27156 The nice value set with *nice()* shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

27158 As -1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *nice()*, and if it returns -1, check to see whether *errno* is non-zero.

27161 **RETURN VALUE**

27162 Upon successful completion, *nice()* shall return the new nice value -{NZERO}. Otherwise, -1 shall be returned, the process' nice value shall not be changed, and *errno* shall be set to indicate the error.

27165 **ERRORS**

27166 The *nice()* function shall fail if:

27167 [EPERM] The *incr* argument is negative and the calling process does not have appropriate privileges.

27169 **EXAMPLES**

27170 **Changing the Nice Value**

27171 The following example adds the value of the *incr* argument, -20, to the nice value of the calling process.

27173 `#include <unistd.h>`

27174 `...`

27175 `int incr = -20;`

27176 `int ret;`

27177 `ret = nice(incr);`

27178 **APPLICATION USAGE**

27179 None.

27180 **RATIONALE**

27181 None.

27182 **FUTURE DIRECTIONS**

27183 None.

27184 **SEE ALSO**

27185 *getpriority()*, *setpriority()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**,
27186 **<unistd.h>**

27187 **CHANGE HISTORY**

27188 First released in Issue 1. Derived from Issue 1 of the SVID.

27189 **Issue 5**

27190 A statement is added to the description indicating the effects of this function on the different
27191 scheduling policies and multi-threaded processes.

27192 **NAME**

27193 nl_langinfo — language information

27194 **SYNOPSIS**

27195 xSI #include <langinfo.h>

27196 char *nl_langinfo(nl_item item);

27197

27198 **DESCRIPTION**

27199 The *nl_langinfo()* function shall return a pointer to a string containing information relevant to
 27200 the particular language or cultural area defined in the program's locale (see <langinfo.h>). The
 27201 manifest constant names and values of *item* are defined in <langinfo.h>. For example:

27202 nl_langinfo(ABDAY_1)

27203 would return a pointer to the string "Dom" if the identified language was Portuguese, and
 27204 "Sun" if the identified language was English.

27205 Calls to *setlocale()* with a category corresponding to the category of *item* (see <langinfo.h>), or to
 27206 the category *LC_ALL*, may overwrite the array pointed to by the return value.

27207 The *nl_langinfo()* function need not be reentrant. A function that is not required to be reentrant is
 27208 not required to be thread-safe.

27209 **RETURN VALUE**

27210 In a locale where *langinfo* data is not defined, *nl_langinfo()* shall return a pointer to the
 27211 corresponding string in the POSIX locale. In all locales, *nl_langinfo()* shall return a pointer to an
 27212 empty string if *item* contains an invalid setting.

27213 This pointer may point to static data that may be overwritten on the next call.

27214 **ERRORS**

27215 No errors are defined.

27216 **EXAMPLES**27217 **Getting Date and Time Formatting Information**

27218 The following example returns a pointer to a string containing date and time formatting
 27219 information, as defined in the *LC_TIME* category of the current locale.

27220 #include <time.h>

27221 #include <langinfo.h>

27222 ...

27223 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);

27224 ...

27225 **APPLICATION USAGE**

27226 The array pointed to by the return value should not be modified by the program, but may be
 27227 modified by further calls to *nl_langinfo()*.

27228 **RATIONALE**

27229 None.

27230 **FUTURE DIRECTIONS**

27231 None.

27232 **SEE ALSO**

27233 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <**langinfo.h**>,
27234 <**nl_types.h**>

27235 **CHANGE HISTORY**

27236 First released in Issue 2.

27237 **Issue 5**

27238 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

27239 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

27240 NAME

27241 nrand48 — generate uniformly distributed pseudo-random non-negative long integers

27242 SYNOPSIS

27243 xSI `#include <stdlib.h>`

27244 `long nrand48(unsigned short xsubi[3]);`

27245

27246 DESCRIPTION

27247 Refer to *drand48()*.

27248 **NAME**

27249 ntohl, ntohs — convert values between host and network byte order

27250 **SYNOPSIS**

27251 #include <arpa/inet.h>

27252 uint32_t ntohl(uint32_t *netlong*);27253 uint16_t ntohs(uint16_t *netshort*);27254 **DESCRIPTION**27255 Refer to *htonl()*.

27256 **NAME**

27257 open — open a file

27258 **SYNOPSIS**

27259 OH #include <sys/stat.h>

27260 #include <fcntl.h>

27261 int open(const char *path, int oflag, ...);

27262 **DESCRIPTION**

27263 The *open()* function shall establish the connection between a file and a file descriptor. It shall
 27264 create an open file description that refers to a file and a file descriptor that refers to that open file
 27265 description. The file descriptor is used by other I/O functions to refer to that file. The *path*
 27266 argument points to a pathname naming the file.

27267 The *open()* function shall return a file descriptor for the named file that is the lowest file
 27268 descriptor not currently open for that process. The open file description is new, and therefore the
 27269 file descriptor shall not share it with any other process in the system. The FD_CLOEXEC file
 27270 descriptor flag associated with the new file descriptor shall be cleared.

27271 The file offset used to mark the current position within the file shall be set to the beginning of the
 27272 file.

27273 The file status flags and file access modes of the open file description shall be set according to
 27274 the value of *oflag*.

27275 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list,
 27276 defined in <fcntl.h>. Applications shall specify exactly one of the first three values (file access
 27277 modes) below in the value of *oflag*:

27278 O_RDONLY Open for reading only.

27279 O_WRONLY Open for writing only.

27280 O_RDWR Open for reading and writing. The result is undefined if this flag is applied to
 27281 a FIFO.

27282 Any combination of the following may be used:

27283 O_APPEND If set, the file offset shall be set to the end of the file prior to each write.

27284 O_CREAT If the file exists, this flag has no effect except as noted under O_EXCL below.
 27285 Otherwise, the file shall be created; the user ID of the file shall be set to the
 27286 effective user ID of the process; the group ID of the file shall be set to the
 27287 group ID of the file's parent directory or to the effective group ID of the
 27288 process; and the access permission bits (see <sys/stat.h>) of the file mode shall
 27289 be set to the value of the third argument taken as type **mode_t** modified as
 27290 follows: a bitwise AND is performed on the file-mode bits and the
 27291 corresponding bits in the complement of the process' file mode creation mask.
 27292 Thus, all bits in the file mode whose corresponding bit in the file mode
 27293 creation mask is set are cleared. When bits other than the file permission bits
 27294 are set, the effect is unspecified. The third argument does not affect whether
 27295 the file is open for reading, writing, or for both. Implementations shall provide
 27296 a way to initialize the file's group ID to the group ID of the parent directory.
 27297 Implementations may, but need not, provide an implementation-defined way
 27298 to initialize the file's group ID to the effective group ID of the calling process.

27299 SIO O_DSYNC Write I/O operations on the file descriptor shall complete as defined by
 27300 synchronized I/O data integrity completion.

27301	O_EXCL	If O_CREAT and O_EXCL are set, <i>open()</i> shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing <i>open()</i> naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and <i>path</i> names a symbolic link, <i>open()</i> shall fail and set <i>errno</i> to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined.
27302		
27303		
27304		
27305		
27306		
27307		
27308	O_NOCTTY	If set and <i>path</i> identifies a terminal device, <i>open()</i> shall not cause the terminal device to become the controlling terminal for the process.
27309		
27310	O_NONBLOCK	When opening a FIFO with O_RDONLY or O_WRONLY set:
27311		• If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return without delay. An <i>open()</i> for writing-only shall return an error if no process currently has the file open for reading.
27312		
27313		
27314		• If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the calling thread until a thread opens the file for writing. An <i>open()</i> for writing-only shall block the calling thread until a thread opens the file for reading.
27315		
27316		
27317		
27318		When opening a block special or character special file that supports non-
27319		blocking opens:
27320		• If O_NONBLOCK is set, the <i>open()</i> function shall return without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.
27321		
27322		
27323		• If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling thread until the device is ready or available before returning.
27324		
27325		Otherwise, the behavior of O_NONBLOCK is unspecified.
27326 SIO	O_RSYNC	Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
27327		
27328		
27329		
27330		
27331		
27332		
27333 SIO	O_SYNC	Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
27334		
27335	O_TRUNC	If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC with O_RDONLY is undefined.
27336		
27337		
27338		
27339		
27340		If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall mark for update the <i>st_atime</i> , <i>st_ctime</i> , and <i>st_mtime</i> fields of the file and the <i>st_ctime</i> and <i>st_mtime</i> fields of the parent directory.
27341		
27342		
27343		If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall mark for update the <i>st_ctime</i> and <i>st_mtime</i> fields of the file.
27344		

27345 SIO	If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.	
27346		
27347 XSR	If <i>path</i> refers to a STREAMS file, <i>oflag</i> may be constructed from O_NONBLOCK OR'ed with	
27348	either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to	
27349	STREAMS devices and shall have no effect on them. The value O_NONBLOCK affects the	
27350	operation of STREAMS drivers and certain functions applied to file descriptors associated with	
27351	STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.	
27352		
27353 XSI	If <i>path</i> names the master side of a pseudo-terminal device, then it is unspecified whether <i>open()</i>	
27354	locks the slave side so that it cannot be opened. Conforming applications shall call <i>unlockpt()</i>	
27355	before opening the slave side.	
27356	The largest value that can be represented correctly in an object of type off_t shall be established	
27357	as the offset maximum in the open file description.	
27358	RETURN VALUE	
27359	Upon successful completion, the function shall open the file and return a non-negative integer	
27360	representing the lowest numbered unused file descriptor. Otherwise, -1 shall be returned and	
27361	<i>errno</i> set to indicate the error. No files shall be created or modified if the function returns -1.	
27362	ERRORS	
27363	The <i>open()</i> function shall fail if:	
27364	[EACCES]	Search permission is denied on a component of the path prefix, or the file
27365		exists and the permissions specified by <i>oflag</i> are denied, or the file does not
27366		exist and write permission is denied for the parent directory of the file to be
27367		created, or O_TRUNC is specified and write permission is denied.
27368	[EEXIST]	O_CREAT and O_EXCL are set, and the named file exists.
27369	[EINTR]	A signal was caught during <i>open()</i> .
27370 SIO	[EINVAL]	The implementation does not support synchronized I/O for this file.
27371 XSR	[EIO]	The <i>path</i> argument names a STREAMS file and a hangup or error occurred
27372		during the <i>open()</i> .
27373	[EISDIR]	The named file is a directory and <i>oflag</i> includes O_WRONLY or O_RDWR.
27374	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i>
27375		argument.
27376	[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
27377	[ENAMETOOLONG]	
27378		The length of the <i>path</i> argument exceeds {PATH_MAX} or a pathname
27379		component is longer than {NAME_MAX}.
27380	[ENFILE]	The maximum allowable number of files is currently open in the system.
27381	[ENOENT]	O_CREAT is not set and the named file does not exist; or O_CREAT is set and
27382		either the path prefix does not exist or the <i>path</i> argument points to an empty
27383		string.
27384 XSR	[ENOSR]	The <i>path</i> argument names a STREAMS-based file and the system is unable to
27385		allocate a STREAM.
27386	[ENOSPC]	The directory or file system that would contain the new file cannot be
27387		expanded, the file does not exist, and O_CREAT is specified.

27388	[ENOTDIR]	A component of the path prefix is not a directory.
27389	[ENXIO]	O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no
27390		process has the file open for reading.
27391	[ENXIO]	The named file is a character special or block special file, and the device
27392		associated with this special file does not exist.
27393	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented
27394		correctly in an object of type off_t .
27395	[EROFS]	The named file resides on a read-only file system and either O_WRONLY,
27396		O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the <i>oflag</i>
27397		argument.
27398	The <i>open()</i> function may fail if:	
27399 XSI	[EAGAIN]	The <i>path</i> argument names the slave side of a pseudo-terminal device that is
27400		locked.
27401	[EINVAL]	The value of the <i>oflag</i> argument is not valid.
27402	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
27403		resolution of the <i>path</i> argument.
27404	[ENAMETOOLONG]	
27405		As a result of encountering a symbolic link in resolution of the <i>path</i> argument,
27406		the length of the substituted pathname string exceeded {PATH_MAX}.
27407 XSR	[ENOMEM]	The <i>path</i> argument names a STREAMS file and the system is unable to allocate
27408		resources.
27409	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>oflag</i> is
27410		O_WRONLY or O_RDWR.

27411 EXAMPLES

27412 Opening a File for Writing by the Owner

27413 The following example opens the file **/tmp/file**, either by creating it (if it does not already exist),
 27414 or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file,
 27415 the access permission bits in the file mode of the file are set to permit reading and writing by the
 27416 owner, and to permit reading only by group members and others.

27417 If the call to *open()* is successful, the file is opened for writing.

```

27418 #include <fcntl.h>
27419 ...
27420 int fd;
27421 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
27422 char *filename = "/tmp/file";
27423 ...
27424 fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
27425 ...

```

Opening a File Using an Existence Check

The following example uses the *open()* function to try to create the **LOCKFILE** file and open it for writing. Since the *open()* function specifies the **O_EXCL** flag, the call fails if the file already exists. In that case, the program assumes that someone else is updating the password file and exits.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd; /* Integer for file descriptor returned by open() call. */
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
```

Opening a File for Writing

The following example opens a file for writing, creating the file if it does not already exist. If the file does exist, the system truncates the file to zero bytes.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd;
char filename[PATH_MAX+1];
...
if ((pfd = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    perror("Cannot open output file\n"); exit(1);
}
...
```

APPLICATION USAGE

None.

RATIONALE

Except as specified in this volume of IEEE Std 1003.1-2001, the flags allowed in *oflag* are not mutually-exclusive and any number of them may be used simultaneously.

Some implementations permit opening FIFOs with **O_RDWR**. Since FIFOs could be implemented in other ways, and since two file descriptors can be used to the same effect, this possibility is left as undefined.

See *getgroups()* about the group of a newly created file.

27471 The use of *open()* to create a regular file is preferable to the use of *creat()*, because the latter is
 27472 redundant and included only for historical reasons.

27473 The use of the *O_TRUNC* flag on FIFOs and directories (pipes cannot be *open()*-ed) must be
 27474 permissible without unexpected side effects (for example, *creat()* on a FIFO must not remove
 27475 data). Since terminal special files might have type-ahead data stored in the buffer, *O_TRUNC*
 27476 should not affect their content, particularly if a program that normally opens a regular file
 27477 should open the current controlling terminal instead. Other file types, particularly
 27478 implementation-defined ones, are left implementation-defined.

27479 IEEE Std 1003.1-2001 permits [*EACCES*] to be returned for conditions other than those explicitly
 27480 listed.

27481 The *O_NOCTTY* flag was added to allow applications to avoid unintentionally acquiring a
 27482 controlling terminal as a side effect of opening a terminal file. This volume of
 27483 IEEE Std 1003.1-2001 does not specify how a controlling terminal is acquired, but it allows an
 27484 implementation to provide this on *open()* if the *O_NOCTTY* flag is not set and other conditions
 27485 specified in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal
 27486 Interface are met. The *O_NOCTTY* flag is an effective no-op if the file being opened is not a
 27487 terminal device.

27488 In historical implementations the value of *O_RDONLY* is zero. Because of that, it is not possible
 27489 to detect the presence of *O_RDONLY* and another option. Future implementations should
 27490 encode *O_RDONLY* and *O_WRONLY* as bit flags so that:

27491 *O_RDONLY* | *O_WRONLY* == *O_RDWR*

27492 In general, the *open()* function follows the symbolic link if *path* names a symbolic link. However,
 27493 the *open()* function, when called with *O_CREAT* and *O_EXCL*, is required to fail with [*EEXIST*]
 27494 if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This
 27495 behavior is required so that privileged applications can create a new file in a known location
 27496 without the possibility that a symbolic link might cause the file to be created in a different
 27497 location.

27498 For example, a privileged application that must create a file with a predictable name in a user-
 27499 writable directory, such as the user's home directory, could be compromised if the user creates a
 27500 symbolic link with that name that refers to a nonexistent file in a system directory. If the user can
 27501 influence the contents of a file, the user could compromise the system by creating a new system
 27502 configuration or spool file that would then be interpreted by the system. The test for a symbolic
 27503 link which refers to a nonexistent file must be atomic with the creation of a new file.

27504 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group
 27505 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required
 27506 that implementations provide a way to have the group ID be set to the group ID of the
 27507 containing directory, but did not prohibit implementations also supporting a way to set the
 27508 group ID to the effective group ID of the creating process. Conforming applications should not
 27509 assume which group ID will be used. If it matters, an application can use *chown()* to set the
 27510 group ID after the file is created, or determine under what conditions the implementation will
 27511 set the desired group ID.

27512 FUTURE DIRECTIONS

27513 None.

27514 SEE ALSO

27515 *chmod()*, *close()*, *creat()*, *dup()*, *fcntl()*, *lseek()*, *read()*, *umask()*, *unlockpt()*, *write()*, the Base
 27516 Definitions volume of IEEE Std 1003.1-2001, <*fcntl.h*>, <*sys/stat.h*>, <*sys/types.h*>

27517 **CHANGE HISTORY**

27518 First released in Issue 1. Derived from Issue 1 of the SVID.

27519 **Issue 5**

27520 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
27521 Threads Extension.

27522 Large File Summit extensions are added.

27523 **Issue 6**

27524 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

27525 The following new requirements on POSIX implementations derive from alignment with the
27526 Single UNIX Specification:

27527 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
27528 required for conforming implementations of previous POSIX specifications, it was not
27529 required for UNIX applications.

27530 • In the DESCRIPTION, O_CREAT is amended to state that the group ID of the file is set to the
27531 group ID of the file's parent directory or to the effective group ID of the process. This is a
27532 FIPS requirement.

27533 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
27534 description. This change is to support large files.

27535 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support
27536 large files.

27537 • The [ENXIO] mandatory error condition is added.

27538 • The [EINVAL], [ENAMETOOLONG], and [ETXTBSY] optional error conditions are added.

27539 The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI
27540 STREAMS Option Group are marked.

27541 The following changes were made to align with the IEEE P1003.1a draft standard:

27542 • An explanation is added of the effect of the O_CREAT and O_EXCL flags when the path
27543 refers to a symbolic link.

27544 • The [ELOOP] optional error condition is added.

27545 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

27546 The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

27547 **NAME**

27548 opendir — open a directory

27549 **SYNOPSIS**

27550 #include <dirent.h>

27551 DIR *opendir(const char *dirname);

27552 **DESCRIPTION**

27553 The *opendir()* function shall open a directory stream corresponding to the directory named by
 27554 the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is
 27555 implemented using a file descriptor, applications shall only be able to open up to a total of
 27556 {OPEN_MAX} files and directories.

27557 **RETURN VALUE**

27558 Upon successful completion, *opendir()* shall return a pointer to an object of type **DIR**.
 27559 Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

27560 **ERRORS**27561 The *opendir()* function shall fail if:

27562 [EACCES] Search permission is denied for the component of the path prefix of *dirname* or
 27563 read permission is denied for *dirname*.

27564 [ELOOP] A loop exists in symbolic links encountered during resolution of the *dirname*
 27565 argument.

27566 [ENAMETOOLONG] The length of the *dirname* argument exceeds {PATH_MAX} or a pathname
 27567 component is longer than {NAME_MAX}.
 27568

27569 [ENOENT] A component of *dirname* does not name an existing directory or *dirname* is an
 27570 empty string.

27571 [ENOTDIR] A component of *dirname* is not a directory.

27572 The *opendir()* function may fail if:

27573 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 27574 resolution of the *dirname* argument.

27575 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

27576 [ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the *dirname*
 27577 argument, the length of the substituted pathname string exceeded
 27578 {PATH_MAX}.
 27579

27580 [ENFILE] Too many files are currently open in the system.

27581 EXAMPLES

27582 Open a Directory Stream

27583 The following program fragment demonstrates how the *opendir()* function is used.

```
27584 #include <sys/types.h>
27585 #include <dirent.h>
27586 #include <libgen.h>
27587 ...
27588     DIR *dir;
27589     struct dirent *dp;
27590 ...
27591     if ((dir = opendir(".")) == NULL) {
27592         perror("Cannot open .");
27593         exit(1);
27594     }
27595     while ((dp = readdir(dir)) != NULL) {
27596         ...
```

27597 APPLICATION USAGE

27598 The *opendir()* function should be used in conjunction with *readdir()*, *closedir()*, and *rewinddir()* to
 27599 examine the contents of the directory (see the EXAMPLES section in *readdir()*). This method is
 27600 recommended for portability.

27601 RATIONALE

27602 Based on historical implementations, the rules about file descriptors apply to directory streams
 27603 as well. However, this volume of IEEE Std 1003.1-2001 does not mandate that the directory
 27604 stream be implemented using file descriptors. The description of *closedir()* clarifies that if a file
 27605 descriptor is used for the directory stream, it is mandatory that *closedir()* deallocate the file
 27606 descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the
 27607 FD_CLOEXEC had been set for the file descriptor.

27608 The directory entries for dot and dot-dot are optional. This volume of IEEE Std 1003.1-2001 does
 27609 not provide a way to test *a priori* for their existence because an application that is portable must
 27610 be written to look for (and usually ignore) those entries. Writing code that presumes that they
 27611 are the first two entries does not always work, as many implementations permit them to be
 27612 other than the first two entries, with a “normal” entry preceding them. There is negligible value
 27613 in providing a way to determine what the implementation does because the code to deal with
 27614 dot and dot-dot must be written in any case and because such a flag would add to the list of
 27615 those flags (which has proven in itself to be objectionable) and might be abused.

27616 Since the structure and buffer allocation, if any, for directory operations are defined by the
 27617 implementation, this volume of IEEE Std 1003.1-2001 imposes no portability requirements for
 27618 erroneous program constructs, erroneous data, or the use of unspecified values such as the use
 27619 or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed
 27620 or after a *fork()* or one of the *exec* function calls.

27621 FUTURE DIRECTIONS

27622 None.

27623 SEE ALSO

27624 *closedir()*, *lstat()*, *readdir()*, *rewinddir()*, *symlink()*, the Base Definitions volume of
 27625 IEEE Std 1003.1-2001, **<dirent.h>**, **<limits.h>**, **<sys/types.h>**

27626 **CHANGE HISTORY**

27627 First released in Issue 2.

27628 **Issue 6**27629 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.27630 The following new requirements on POSIX implementations derive from alignment with the
27631 Single UNIX Specification:27632 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
27633 required for conforming implementations of previous POSIX specifications, it was not
27634 required for UNIX applications.

27635 • The [ELOOP] mandatory error condition is added.

27636 • A second [ENAMETOOLONG] is added as an optional error condition.

27637 The following changes were made to align with the IEEE P1003.1a draft standard:

27638 • The [ELOOP] optional error condition is added.

27639 NAME

27640 openlog — open a connection to the logging facility

27641 SYNOPSIS

27642 xSI #include <syslog.h>

27643 void openlog(const char *ident, int logopt, int facility);

27644

27645 DESCRIPTION

27646 Refer to *closelog()*.

27647 **NAME**

27648 optarg, opterr, optind, optopt — options parsing variables

27649 **SYNOPSIS**

27650 #include <unistd.h>

27651 extern char *optarg;

27652 extern int opterr, optind, optopt;

27653 **DESCRIPTION**

27654 Refer to *getopt()*.

27655 **NAME**

27656 pathconf — get configurable pathname variables

27657 **SYNOPSIS**

27658 #include <unistd.h>

27659 long pathconf(const char *path, int name);

27660 **DESCRIPTION**

27661 Refer to *fpathconf()*.

27662 **NAME**

27663 *pause* — suspend the thread until a signal is received

27664 **SYNOPSIS**

27665 `#include <unistd.h>`

27666 `int pause(void);`

27667 **DESCRIPTION**

27668 The *pause()* function shall suspend the calling thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

27670 If the action is to terminate the process, *pause()* shall not return.

27671 If the action is to execute a signal-catching function, *pause()* shall return after the signal-catching function returns.

27673 **RETURN VALUE**

27674 Since *pause()* suspends thread execution indefinitely unless interrupted by a signal, there is no successful completion return value. A value of `-1` shall be returned and *errno* set to indicate the error.

27677 **ERRORS**

27678 The *pause()* function shall fail if:

27679 [EINTR] A signal is caught by the calling process and control is returned from the
27680 signal-catching function.

27681 **EXAMPLES**

27682 None.

27683 **APPLICATION USAGE**

27684 Many common uses of *pause()* have timing windows. The scenario involves checking a
27685 condition related to a signal and, if the signal has not occurred, calling *pause()*. When the signal
27686 occurs between the check and the call to *pause()*, the process often blocks indefinitely. The
27687 *sigprocmask()* and *sigsuspend()* functions can be used to avoid this type of problem.

27688 **RATIONALE**

27689 None.

27690 **FUTURE DIRECTIONS**

27691 None.

27692 **SEE ALSO**

27693 *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<unistd.h>`

27694 **CHANGE HISTORY**

27695 First released in Issue 1. Derived from Issue 1 of the SVID.

27696 **Issue 5**

27697 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

27698 **Issue 6**

27699 The APPLICATION USAGE section is added.

27700 **NAME**

27701 pclose — close a pipe stream to or from a process

27702 **SYNOPSIS**27703 `CX` #include <stdio.h>

27704 int pclose(FILE *stream);

27705

27706 **DESCRIPTION**

27707 The *pclose()* function shall close a stream that was opened by *popen()*, wait for the command to
 27708 terminate, and return the termination status of the process that was running the command
 27709 language interpreter. However, if a call caused the termination status to be unavailable to
 27710 *pclose()*, then *pclose()* shall return -1 with *errno* set to [ECHILD] to report this situation. This can
 27711 happen if the application calls one of the following functions:

- 27712 • *wait()*
- 27713 • *waitpid()* with a *pid* argument less than or equal to 0 or equal to the process ID of the
 27714 command line interpreter
- 27715 • Any other function not defined in this volume of IEEE Std 1003.1-2001 that could do one of
 27716 the above

27717 In any case, *pclose()* shall not return before the child process created by *popen()* has terminated.

27718 If the command language interpreter cannot be executed, the child termination status returned
 27719 by *pclose()* shall be as if the command language interpreter terminated using *exit(127)* or
 27720 *_exit(127)*.

27721 The *pclose()* function shall not affect the termination status of any child of the calling process
 27722 other than the one created by *popen()* for the associated stream.

27723 If the argument *stream* to *pclose()* is not a pointer to a stream created by *popen()*, the result of
 27724 *pclose()* is undefined.

27725 **RETURN VALUE**

27726 Upon successful return, *pclose()* shall return the termination status of the command language
 27727 interpreter. Otherwise, *pclose()* shall return -1 and set *errno* to indicate the error.

27728 **ERRORS**

27729 The *pclose()* function shall fail if:

27730 [ECHILD] The status of the child process could not be obtained, as described above.

27731 **EXAMPLES**

27732 None.

27733 **APPLICATION USAGE**

27734 None.

27735 **RATIONALE**

27736 There is a requirement that *pclose()* not return before the child process terminates. This is
 27737 intended to disallow implementations that return [EINTR] if a signal is received while waiting.
 27738 If *pclose()* returned before the child terminated, there would be no way for the application to
 27739 discover which child used to be associated with the stream, and it could not do the cleanup
 27740 itself.

27741 If the stream pointed to by *stream* was not created by *popen()*, historical implementations of
 27742 *pclose()* return -1 without setting *errno*. To avoid requiring *pclose()* to set *errno* in this case,
 27743 IEEE Std 1003.1-2001 makes the behavior unspecified. An application should not use *pclose()* to

27744 close any stream that was not created by *popen()*.

27745 Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT,
27746 and SIGHUP while waiting for the child process to terminate. Since this behavior is not
27747 described for the *pclose()* function in IEEE Std 1003.1-2001, such implementations are not
27748 conforming. Also, some historical implementations return [EINTR] if a signal is received, even
27749 though the child process has not terminated. Such implementations are also considered non-
27750 conforming.

27751 Consider, for example, an application that uses:

```
27752 popen("command", "r")
```

27753 to start *command*, which is part of the same application. The parent writes a prompt to its
27754 standard output (presumably the terminal) and then reads from the *popen()*ed stream. The child
27755 reads the response from the user, does some transformation on the response (pathname
27756 expansion, perhaps) and writes the result to its standard output. The parent process reads the
27757 result from the pipe, does something with it, and prints another prompt. The cycle repeats.
27758 Assuming that both processes do appropriate buffer flushing, this would be expected to work.

27759 To conform to IEEE Std 1003.1-2001, *pclose()* must use *waitpid()*, or some similar function,
27760 instead of *wait()*.

27761 The code sample below illustrates how the *pclose()* function might be implemented on a system
27762 conforming to IEEE Std 1003.1-2001.

```
27763 int pclose(FILE *stream)
27764 {
27765     int stat;
27766     pid_t pid;
27767
27768     pid = <pid for process created for stream by popen()>
27769     (void) fclose(stream);
27770     while (waitpid(pid, &stat, 0) == -1) {
27771         if (errno != EINTR) {
27772             stat = -1;
27773             break;
27774         }
27775     }
27776     return(stat);
27777 }
```

27777 FUTURE DIRECTIONS

27778 None.

27779 SEE ALSO

27780 *fork()*, *popen()*, *waitpid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

27781 CHANGE HISTORY

27782 First released in Issue 1. Derived from Issue 1 of the SVID.

27783 **NAME**

27784 perror — write error messages to standard error

27785 **SYNOPSIS**

27786 #include <stdio.h>

27787 void perror(const char *s);

27788 **DESCRIPTION**

27789 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27790 conflict between the requirements described here and the ISO C standard is unintentional. This
 27791 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

27792 The *perror()* function shall map the error number accessed through the symbol *errno* to a
 27793 language-dependent error message, which shall be written to the standard error stream as
 27794 follows:

- 27795 • First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the string
 27796 pointed to by *s* followed by a colon and a <space>.
- 27797 • Then an error message string followed by a <newline>.

27798 The contents of the error message strings shall be the same as those returned by *strerror()* with
 27799 argument *errno*.

27800 CX The *perror()* function shall mark the file associated with the standard error stream as having
 27801 been written (*st_ctime*, *st_mtime* marked for update) at some time between its successful
 27802 completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()* on *stderr*.

27803 The *perror()* function shall not change the orientation of the standard error stream.

27804 **RETURN VALUE**27805 The *perror()* function shall not return a value.27806 **ERRORS**

27807 No errors are defined.

27808 **EXAMPLES**27809 **Printing an Error Message for a Function**

27810 The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs,
 27811 the *perror()* function prints a message and the program exits.

```

27812     #include <stdio.h>
27813     #include <stdlib.h>
27814     ...
27815     char *bufptr;
27816     size_t szbuf;
27817     ...
27818     if ((bufptr = malloc(szbuf)) == NULL) {
27819         perror("malloc"); exit(2);
27820     }
27821     ...
```

27822 **APPLICATION USAGE**

27823 None.

27824 **RATIONALE**

27825 None.

27826 **FUTURE DIRECTIONS**

27827 None.

27828 **SEE ALSO**27829 *strerror()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>27830 **CHANGE HISTORY**

27831 First released in Issue 1. Derived from Issue 1 of the SVID.

27832 **Issue 5**27833 A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the
27834 orientation of the standard error stream.27835 **Issue 6**

27836 Extensions beyond the ISO C standard are marked.

27837 **NAME**

27838 pipe — create an interprocess channel

27839 **SYNOPSIS**

27840 #include <unistd.h>

27841 int pipe(int *fildes*[2]);27842 **DESCRIPTION**

27843 The *pipe()* function shall create a pipe and place two file descriptors, one each into the
 27844 arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write
 27845 ends of the pipe. Their integer values shall be the two lowest available at the time of the *pipe()*
 27846 call. The O_NONBLOCK and FD_CLOEXEC flags shall be clear on both file descriptors. (The
 27847 *fcntl()* function can be used to set both these flags.)

27848 Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A
 27849 read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a
 27850 first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether
 27851 *fildes*[1] is also open for reading.

27852 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open
 27853 that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

27854 Upon successful completion, *pipe()* shall mark for update the *st_atime*, *st_ctime*, and *st_mtime*
 27855 fields of the pipe.

27856 **RETURN VALUE**

27857 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 27858 indicate the error.

27859 **ERRORS**27860 The *pipe()* function shall fail if:

27861	[EMFILE]	More than {OPEN_MAX} minus two file descriptors are already in use by this process.
27862		

27863	[ENFILE]	The number of simultaneously open files in the system would exceed a system-imposed limit.
27864		

27865 **EXAMPLES**

27866 None.

27867 **APPLICATION USAGE**

27868 None.

27869 **RATIONALE**

27870 The wording carefully avoids using the verb “to open” in order to avoid any implication of use
 27871 of *open()*; see also *write()*.

27872 **FUTURE DIRECTIONS**

27873 None.

27874 **SEE ALSO**

27875 *fcntl()*, *read()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <*fcntl.h*>,
 27876 <*unistd.h*>

27877 **CHANGE HISTORY**

27878 First released in Issue 1. Derived from Issue 1 of the SVID.

27879 **Issue 6**

27880 The following new requirements on POSIX implementations derive from alignment with the
27881 Single UNIX Specification:

- 27882 • The DESCRIPTION is updated to indicate that certain dispositions of *fildev[0]* and *fildev[1]*
27883 are unspecified.

27884 **NAME**

27885 poll — input/output multiplexing

27886 **SYNOPSIS**27887 xSI `#include <poll.h>`27888 `int poll(struct pollfd fds[], nfds_t nfds, int timeout);`

27889

27890 **DESCRIPTION**

27891 The `poll()` function provides applications with a mechanism for multiplexing input/output over
 27892 a set of file descriptors. For each member of the array pointed to by `fds`, `poll()` shall examine the
 27893 given file descriptor for the event(s) specified in `events`. The number of **pollfd** structures in the
 27894 `fds` array is specified by `nfds`. The `poll()` function shall identify those file descriptors on which an
 27895 application can read or write data, or on which certain events have occurred.

27896 The `fds` argument specifies the file descriptors to be examined and the events of interest for each
 27897 file descriptor. It is a pointer to an array with one member for each open file descriptor of
 27898 interest. The array's members are **pollfd** structures within which `fd` specifies an open file
 27899 descriptor and `events` and `revents` are bitmasks constructed by OR'ing a combination of the
 27900 following event flags:

27901	POLLIN	Data other than high-priority data may be read without blocking.
27902 xSR		For STREAMS, this flag is set in <code>revents</code> even if the message is of zero length.
27903		This flag shall be equivalent to <code>POLLRDNORM POLLRDBAND</code> .
27904	POLLRDNORM	Normal data may be read without blocking.
27905 xSR		For STREAMS, data on priority band 0 may be read without blocking. This
27906		flag is set in <code>revents</code> even if the message is of zero length.
27907	POLLRDBAND	Priority data may be read without blocking.
27908 xSR		For STREAMS, data on priority bands greater than 0 may be read without
27909		blocking. This flag is set in <code>revents</code> even if the message is of zero length.
27910	POLLPRI	High-priority data may be read without blocking.
27911 xSR		For STREAMS, this flag is set in <code>revents</code> even if the message is of zero length.
27912	POLLOUT	Normal data may be written without blocking.
27913 xSR		For STREAMS, data on priority band 0 may be written without blocking.
27914	POLLWRNORM	Equivalent to <code>POLLOUT</code> .
27915	POLLWRBAND	Priority data may be written.
27916 xSR		For STREAMS, data on priority bands greater than 0 may be written without
27917		blocking. If any priority band has been written to on this STREAM, this event
27918		only examines bands that have been written to at least once.
27919	POLLERR	An error has occurred on the device or stream. This flag is only valid in the
27920		<code>revents</code> bitmask; it shall be ignored in the <code>events</code> member.
27921	POLLHUP	The device has been disconnected. This event and <code>POLLOUT</code> are mutually-
27922		exclusive; a stream can never be writable if a hangup has occurred. However,
27923		this event and <code>POLLIN</code> , <code>POLLRDNORM</code> , <code>POLLRDBAND</code> , or <code>POLLPRI</code> are not
27924		mutually-exclusive. This flag is only valid in the <code>revents</code> bitmask; it shall be
27925		ignored in the <code>events</code> member.

27926	POLLNVAL	The specified <i>fd</i> value is invalid. This flag is only valid in the <i>revents</i> member; it shall ignored in the <i>events</i> member.
27927		
27928		The significance and semantics of normal, priority, and high-priority data are file and device-specific.
27929		
27930		If the value of <i>fd</i> is less than 0, <i>events</i> shall be ignored, and <i>revents</i> shall be set to 0 in that entry on return from <i>poll()</i> .
27931		
27932		In each pollfd structure, <i>poll()</i> shall clear the <i>revents</i> member, except that where the application requested a report on a condition by setting one of the bits of <i>events</i> listed above, <i>poll()</i> shall set the corresponding bit in <i>revents</i> if the requested condition is true. In addition, <i>poll()</i> shall set the POLLHUP, POLLERR, and POLLNVAL flag in <i>revents</i> if the condition is true, even if the application did not set the corresponding bit in <i>events</i> .
27933		
27934		
27935		
27936		
27937		If none of the defined events have occurred on any selected file descriptor, <i>poll()</i> shall wait at least <i>timeout</i> milliseconds for an event to occur on any of the selected file descriptors. If the value of <i>timeout</i> is 0, <i>poll()</i> shall return immediately. If the value of <i>timeout</i> is -1, <i>poll()</i> shall block until a requested event occurs or until the call is interrupted.
27938		
27939		
27940		
27941		Implementations may place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.
27942		
27943		
27944		The <i>poll()</i> function shall not be affected by the O_NONBLOCK flag.
27945		The <i>poll()</i> function shall support regular files, terminal and pseudo-terminal devices, FIFOs, pipes, sockets and STREAMS-based files. The behavior of <i>poll()</i> on elements of <i>fds</i> that refer to other types of file is unspecified.
27946	XSR	
27947		
27948		Regular files shall always poll TRUE for reading and writing.
27949		A file descriptor for a socket that is listening for connections shall indicate that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously shall indicate that it is ready for writing, once a connection has been established.
27950		
27951		
27952	RETURN VALUE	
27953		Upon successful completion, <i>poll()</i> shall return a non-negative value. A positive value indicates the total number of file descriptors that have been selected (that is, file descriptors for which the <i>revents</i> member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, <i>poll()</i> shall return -1 and set <i>errno</i> to indicate the error.
27954		
27955		
27956		
27957	ERRORS	
27958		The <i>poll()</i> function shall fail if:
27959	[EAGAIN]	The allocation of internal data structures failed but a subsequent request may succeed.
27960		
27961	[EINTR]	A signal was caught during <i>poll()</i> .
27962	XSR	
27963	[EINVAL]	The <i>nfds</i> argument is greater than {OPEN_MAX}, or one of the <i>fd</i> members refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.
27964		

27965 **EXAMPLES**27966 **Checking for Events on a Stream**

27967 The following example opens a pair of STREAMS devices and then waits for either one to
 27968 become writable. This example proceeds as follows:

- 27969 1. Sets the *timeout* parameter to 500 milliseconds.
- 27970 2. Opens the STREAMS devices **/dev/dev0** and **/dev/dev1**, and then polls them, specifying
 27971 POLLOUT and POLLWRBAND as the events of interest.
- 27972 The STREAMS device names **/dev/dev0** and **/dev/dev1** are only examples of how
 27973 STREAMS devices can be named; STREAMS naming conventions may vary among
 27974 systems conforming to the IEEE Std 1003.1-2001.
- 27975 3. Uses the *ret* variable to determine whether an event has occurred on either of the two
 27976 STREAMS. The *poll()* function is given 500 milliseconds to wait for an event to occur (if it
 27977 has not occurred prior to the *poll()* call).
- 27978 4. Checks the returned value of *ret*. If a positive value is returned, one of the following can
 27979 be done:
 - 27980 a. Priority data can be written to the open STREAM on priority bands greater than 0,
 27981 because the POLLWRBAND event occurred on the open STREAM (*fds*[0] or *fds*[1]).
 - 27982 b. Data can be written to the open STREAM on priority-band 0, because the POLLOUT
 27983 event occurred on the open STREAM (*fds*[0] or *fds*[1]).
- 27984 5. If the returned value is not a positive value, permission to write data to the open STREAM
 27985 (on any priority band) is denied.
- 27986 6. If the POLLHUP event occurs on the open STREAM (*fds*[0] or *fds*[1]), the device on the
 27987 open STREAM has disconnected.

```

27988 #include <stropts.h>
27989 #include <poll.h>
27990 ...
27991 struct pollfd fds[2];
27992 int timeout_msecs = 500;
27993 int ret;
27994 int i;

27995 /* Open STREAMS device. */
27996 fds[0].fd = open("/dev/dev0", ...);
27997 fds[1].fd = open("/dev/dev1", ...);
27998     fds[0].events = POLLOUT | POLLWRBAND;
27999     fds[1].events = POLLOUT | POLLWRBAND;

28000 ret = poll(fds, 2, timeout_msecs);

28001 if (ret > 0) {
28002     /* An event on one of the fds has occurred. */
28003     for (i=0; i<2; i++) {
28004         if (fds[i].revents & POLLWRBAND) {
28005             /* Priority data may be written on device number i. */
28006             ...
28007         }
28008         if (fds[i].revents & POLLOUT) {

```

```

28009          /* Data may be written on device number i. */
28010      ...
28011      }
28012      if (fds[i].revents & POLLHUP) {
28013          /* A hangup has occurred on device number i. */
28014      ...
28015      }
28016      }
28017  }

```

28018 APPLICATION USAGE

28019 None.

28020 RATIONALE

28021 None.

28022 FUTURE DIRECTIONS

28023 None.

28024 SEE ALSO

28025 Section 2.6 (on page 38), *getmsg()*, *putmsg()*, *read()*, *select()*, *write()*, the Base Definitions volume
 28026 of IEEE Std 1003.1-2001, <**poll.h**>, <**stropts.h**>

28027 CHANGE HISTORY

28028 First released in Issue 4, Version 2.

28029 Issue 5

28030 Moved from X/OPEN UNIX extension to BASE.

28031 The description of POLLWRBAND is updated.

28032 Issue 6

28033 Text referring to sockets is added to the DESCRIPTION.

28034 Text relating to the XSI STREAMS Option Group is marked.

28035 The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of
 28036 POLLWRBAND.

28037 **NAME**

28038 popen — initiate pipe streams to or from a process

28039 **SYNOPSIS**28040 CX `#include <stdio.h>`28041 `FILE *popen(const char *command, const char *mode);`

28042

28043 **DESCRIPTION**

28044 The *popen()* function shall execute the command specified by the string *command*. It shall create a
 28045 pipe between the calling program and the executed command, and shall return a pointer to a
 28046 stream that can be used to either read from or write to the pipe.

28047 The environment of the executed command shall be as if a child process were created within the
 28048 *popen()* call using the *fork()* function, and the child invoked the *sh* utility using the call:

28049 `execl(shell_path, "sh", "-c", command, (char *)0);`28050 where *shell_path* is an unspecified pathname for the *sh* utility.

28051 The *popen()* function shall ensure that any streams from previous *popen()* calls that remain open
 28052 in the parent process are closed in the new child process.

28053 The *mode* argument to *popen()* is a string that specifies I/O mode:

- 28054 1. If *mode* is *r*, when the child process is started, its file descriptor `STDOUT_FILENO` shall be
 28055 the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,
 28056 where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the
 28057 pipe.
- 28058 2. If *mode* is *w*, when the child process is started its file descriptor `STDIN_FILENO` shall be
 28059 the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,
 28060 where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the
 28061 pipe.
- 28062 3. If *mode* is any other value, the result is undefined.

28063 After *popen()*, both the parent and the child process shall be capable of executing independently
 28064 before either terminates.

28065 Pipe streams are byte-oriented.

28066 **RETURN VALUE**

28067 Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to
 28068 read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate
 28069 the error.

28070 **ERRORS**28071 The *popen()* function may fail if:

28072 [EMFILE] {FOPEN_MAX} or {STREAM_MAX} streams are currently open in the calling
 28073 process.

28074 [EINVAL] The *mode* argument is invalid.28075 The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

28076 **EXAMPLES**

28077 None.

28078 **APPLICATION USAGE**

28079 Since open files are shared, a mode *r* command can be used as an input filter and a mode *w*
 28080 command as an output filter.

28081 Buffered reading before opening an input filter may leave the standard input of that filter
 28082 mispositioned. Similar problems with an output filter may be prevented by careful buffer
 28083 flushing; for example, with *fflush()*.

28084 A stream opened by *popen()* should be closed by *pclose()*.

28085 The behavior of *popen()* is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and
 28086 *wb* might be supported by specific implementations, but these would not be portable features.
 28087 Note that historical implementations of *popen()* only check to see if the first character of *mode* is
 28088 *r*. Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be
 28089 treated as *mode w*.

28090 If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a
 28091 stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process
 28092 started by *popen()*.

28093 To determine whether or not the environment specified in the Shell and Utilities volume of
 28094 IEEE Std 1003.1-2001 is present, use the function call:

28095 *sysconf* (*_SC_2_VERSION*)

28096 (See *sysconf()*).

28097 **RATIONALE**

28098 The *popen()* function should not be used by programs that have set user (or group) ID privileges.
 28099 The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead.
 28100 This prevents any unforeseen manipulation of the environment of the user that could cause
 28101 execution of commands not anticipated by the calling program.

28102 If the original and *popen()*ed processes both intend to read or write or read and write a common
 28103 file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for
 28104 sharing file handles must be observed (see Section 2.5.1 (on page 35)).

28105 **FUTURE DIRECTIONS**

28106 None.

28107 **SEE ALSO**

28108 *pclose()*, *pipe()*, *sysconf()*, *system()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 28109 <**stdio.h**>, the Shell and Utilities volume of IEEE Std 1003.1-2001, *sh*

28110 **CHANGE HISTORY**

28111 First released in Issue 1. Derived from Issue 1 of the SVID.

28112 **Issue 5**

28113 A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

28114 **Issue 6**

28115 The following new requirements on POSIX implementations derive from alignment with the
 28116 Single UNIX Specification:

- 28117 • The optional [EMFILE] error condition is added.

28118 NAME

28119 posix_fadvise — file advisory information (**ADVANCED REALTIME**)

28120 SYNOPSIS

28121 ADV `#include <fcntl.h>`

28122 `int posix_fadvise(int fd, off_t offset, size_t len, int advice);`

28123

28124 DESCRIPTION

28125 The *posix_fadvise()* function shall advise the implementation on the expected behavior of the
 28126 application with respect to the data in the file associated with the open file descriptor, *fd*,
 28127 starting at *offset* and continuing for *len* bytes. The specified range need not currently exist in the
 28128 file. If *len* is zero, all data following *offset* is specified. The implementation may use this
 28129 information to optimize handling of the specified data. The *posix_fadvise()* function shall have no
 28130 effect on the semantics of other operations on the specified data, although it may affect the
 28131 performance of other operations.

28132 The advice to be applied to the data is specified by the *advice* parameter and may be one of the
 28133 following values:

28134 POSIX_FADV_NORMAL

28135 Specifies that the application has no advice to give on its behavior with respect to the
 28136 specified data. It is the default characteristic if no advice is given for an open file.

28137 POSIX_FADV_SEQUENTIAL

28138 Specifies that the application expects to access the specified data sequentially from lower
 28139 offsets to higher offsets.

28140 POSIX_FADV_RANDOM

28141 Specifies that the application expects to access the specified data in a random order.

28142 POSIX_FADV_WILLNEED

28143 Specifies that the application expects to access the specified data in the near future.

28144 POSIX_FADV_DONTNEED

28145 Specifies that the application expects that it will not access the specified data in the near
 28146 future.

28147 POSIX_FADV_NOREUSE

28148 Specifies that the application expects to access the specified data once and then not reuse it
 28149 thereafter.

28150 These values are defined in `<fcntl.h>`.

28151 RETURN VALUE

28152 Upon successful completion, *posix_fadvise()* shall return zero; otherwise, an error number shall
 28153 be returned to indicate the error.

28154 ERRORS

28155 The *posix_fadvise()* function shall fail if:

28156 [EBADF] The *fd* argument is not a valid file descriptor.

28157 [EINVAL] The value of *advice* is invalid.

28158 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

28159 **EXAMPLES**

28160 None.

28161 **APPLICATION USAGE**

28162 The *posix_fadvise()* function is part of the Advisory Information option and need not be provided
28163 on all implementations.

28164 **RATIONALE**

28165 None.

28166 **FUTURE DIRECTIONS**

28167 None.

28168 **SEE ALSO**28169 *posix_madvise()*, the Base Definitions volume of IEEE Std 1003.1-2001, <fcntl.h>28170 **CHANGE HISTORY**

28171 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28172 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

28173 NAME

28174 posix_fallocate — file space control (**ADVANCED REALTIME**)

28175 SYNOPSIS

28176 ADV `#include <fcntl.h>`

28177 `int posix_fallocate(int fd, off_t offset, size_t len);`

28178

28179 DESCRIPTION

28180 The *posix_fallocate()* function shall ensure that any required storage for regular file data starting
28181 at *offset* and continuing for *len* bytes is allocated on the file system storage media. If
28182 *posix_fallocate()* returns successfully, subsequent writes to the specified file data shall not fail
28183 due to the lack of free space on the file system storage media.

28184 If the *offset+len* is beyond the current file size, then *posix_fallocate()* shall adjust the file size to
28185 *offset+len*. Otherwise, the file size shall not be changed.

28186 It is implementation-defined whether a previous *posix_fadvise()* call influences allocation
28187 strategy.

28188 Space allocated via *posix_fallocate()* shall be freed by a successful call to *creat()* or *open()* that
28189 truncates the size of the file. Space allocated via *posix_fallocate()* may be freed by a successful call
28190 to *ftruncate()* that reduces the file size to a size smaller than *offset+len*.

28191 RETURN VALUE

28192 Upon successful completion, *posix_fallocate()* shall return zero; otherwise, an error number shall
28193 be returned to indicate the error.

28194 ERRORS

28195 The *posix_fallocate()* function shall fail if:

- 28196 [EBADF] The *fd* argument is not a valid file descriptor.
- 28197 [EBADF] The *fd* argument references a file that was opened without write permission.
- 28198 [EFBIG] The value of *offset+len* is greater than the maximum file size.
- 28199 [EINTR] A signal was caught during execution.
- 28200 [EINVAL] The *len* argument was zero or the *offset* argument was less than zero.
- 28201 [EIO] An I/O error occurred while reading from or writing to a file system.
- 28202 [ENODEV] The *fd* argument does not refer to a regular file.
- 28203 [ENOSPC] There is insufficient free space remaining on the file system storage media.
- 28204 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

28205 EXAMPLES

28206 None.

28207 APPLICATION USAGE

28208 The *posix_fallocate()* function is part of the Advisory Information option and need not be
28209 provided on all implementations.

28210 RATIONALE

28211 None.

28212 **FUTURE DIRECTIONS**

28213 None.

28214 **SEE ALSO**

28215 *creat()*, *ftruncate()*, *open()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001,
28216 <fcntl.h>

28217 **CHANGE HISTORY**

28218 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28219 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

28220 NAME

28221 posix_madvise — memory advisory information and alignment control (**ADVANCED**
28222 **REALTIME**)

28223 SYNOPSIS

28224 ADV `#include <sys/mman.h>`

28225 `int posix_madvise(void *addr, size_t len, int advice);`
28226

28227 DESCRIPTION

28228 MF|SHM The *posix_madvise()* function need only be supported if either the Memory Mapped Files or the
28229 Shared Memory Objects options are supported.

28230 The *posix_madvise()* function shall advise the implementation on the expected behavior of the
28231 application with respect to the data in the memory starting at address *addr*, and continuing for
28232 *len* bytes. The implementation may use this information to optimize handling of the specified
28233 data. The *posix_madvise()* function shall have no effect on the semantics of access to memory in
28234 the specified range, although it may affect the performance of access.

28235 The implementation may require that *addr* be a multiple of the page size, which is the value
28236 returned by *sysconf()* when the name value *_SC_PAGESIZE* is used.

28237 The advice to be applied to the memory range is specified by the *advice* parameter and may be
28238 one of the following values:

28239 POSIX_MADV_NORMAL

28240 Specifies that the application has no advice to give on its behavior with respect to the
28241 specified range. It is the default characteristic if no advice is given for a range of memory.

28242 POSIX_MADV_SEQUENTIAL

28243 Specifies that the application expects to access the specified range sequentially from lower
28244 addresses to higher addresses.

28245 POSIX_MADV_RANDOM

28246 Specifies that the application expects to access the specified range in a random order.

28247 POSIX_MADV_WILLNEED

28248 Specifies that the application expects to access the specified range in the near future.

28249 POSIX_MADV_DONTNEED

28250 Specifies that the application expects that it will not access the specified range in the near
28251 future.

28252 These values are defined in the `<sys/mman.h>` header.

28253 RETURN VALUE

28254 Upon successful completion, *posix_madvise()* shall return zero; otherwise, an error number shall
28255 be returned to indicate the error.

28256 ERRORS

28257 The *posix_madvise()* function shall fail if:

28258 [EINVAL] The value of *advice* is invalid.

28259 [ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly
28260 or completely outside the range allowed for the address space of the calling
28261 process.

28262 The *posix_madvise()* function may fail if:

28263 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the
28264 name value *_SC_PAGESIZE* is used.

28265 [EINVAL] The value of *len* is zero.

28266 **EXAMPLES**

28267 None.

28268 **APPLICATION USAGE**

28269 The *posix_madvise()* function is part of the Advisory Information option and need not be
28270 provided on all implementations.

28271 **RATIONALE**

28272 None.

28273 **FUTURE DIRECTIONS**

28274 None.

28275 **SEE ALSO**

28276 *mmap()*, *posix_fadvise()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001,
28277 **<sys/mman.h>**

28278 **CHANGE HISTORY**

28279 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28280 IEEE PASC Interpretation 1003.1 #102 is applied.

28281 NAME

28282 `posix_mem_offset` — find offset and length of a mapped typed memory block (**ADVANCED**
28283 **REALTIME**)

28284 SYNOPSIS

28285 TYM `#include <sys/mman.h>`

```
28286        int posix_mem_offset(const void *restrict addr, size_t len,
28287                            off_t *restrict off, size_t *restrict contig_len,
28288                            int *restrict fildes);
28289
```

28290 DESCRIPTION

28291 The `posix_mem_offset()` function shall return in the variable pointed to by *off* a value that
28292 identifies the offset (or location), within a memory object, of the memory block currently
28293 mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used
28294 (via `mmap()`) to establish the mapping which contains *addr*. If that descriptor was closed since
28295 the mapping was established, the returned value of *fildes* shall be `-1`. The *len* argument specifies
28296 the length of the block of the memory object the user wishes the offset for; upon return, the value
28297 pointed to by *contig_len* shall equal either *len*, or the length of the largest contiguous block of the
28298 memory object that is currently mapped to the calling process starting at *addr*, whichever is
28299 smaller.

28300 If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig_len*
28301 values obtained by calling `posix_mem_offset()` are used in a call to `mmap()` with a file descriptor
28302 that refers to the same memory pool as *fildes* (either through the same port or through a different
28303 port), and that was opened with neither the `POSIX_TYPED_MEM_ALLOCATE` nor the
28304 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag, the typed memory area that is mapped shall
28305 be exactly the same area that was mapped at *addr* in the address space of the process that called
28306 `posix_mem_offset()`.

28307 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this
28308 function is implementation-defined.

28309 RETURN VALUE

28310 Upon successful completion, the `posix_mem_offset()` function shall return zero; otherwise, the
28311 corresponding error status value shall be returned.

28312 ERRORS

28313 The `posix_mem_offset()` function shall fail if:

28314 [EACCES] The process has not mapped a memory object supported by this function at
28315 the given address *addr*.

28316 This function shall not return an error code of [EINTR].

28317 EXAMPLES

28318 None.

28319 APPLICATION USAGE

28320 None.

28321 RATIONALE

28322 None.

28323 **FUTURE DIRECTIONS**

28324 None.

28325 **SEE ALSO**

28326 *mmap()*, *posix_typed_mem_open()*, the Base Definitions volume of IEEE Std 1003.1-2001,
28327 <**sys/mman.h**>

28328 **CHANGE HISTORY**

28329 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

28330 NAME

28331 posix_memalign — aligned memory allocation (**ADVANCED REALTIME**)

28332 SYNOPSIS

28333 ADV `#include <stdlib.h>`

28334 `int posix_memalign(void **memptr, size_t alignment, size_t size);`

28335

28336 DESCRIPTION

28337 The *posix_memalign()* function shall allocate *size* bytes aligned on a boundary specified by
 28338 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*
 28339 shall be a multiple of *sizeof(void *)*, that is also a power of two. Upon successful completion, the
 28340 value pointed to by *memptr* shall be a multiple of *alignment*.

28341 CX The *free()* function shall deallocate memory that has previously been allocated by
 28342 *posix_memalign()*.

28343 RETURN VALUE

28344 Upon successful completion, *posix_memalign()* shall return zero; otherwise, an error number
 28345 shall be returned to indicate the error.

28346 ERRORS

28347 The *posix_memalign()* function shall fail if:

28348 [EINVAL] The value of the alignment parameter is not a power of two multiple of
 28349 *sizeof(void *)*.

28350 [ENOMEM] There is insufficient memory available with the requested alignment.

28351 EXAMPLES

28352 None.

28353 APPLICATION USAGE

28354 The *posix_memalign()* function is part of the Advisory Information option and need not be
 28355 provided on all implementations.

28356 RATIONALE

28357 None.

28358 FUTURE DIRECTIONS

28359 None.

28360 SEE ALSO

28361 *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

28362 CHANGE HISTORY

28363 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28364 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

28365 **NAME**

28366 posix_openpt — open a pseudo-terminal device

28367 **SYNOPSIS**

28368 xSI #include <stdlib.h>

28369 #include <fcntl.h>

28370 int posix_openpt(int oflag);

28371

28372 **DESCRIPTION**

28373 The *posix_openpt()* function shall establish a connection between a master device for a pseudo-terminal and a file descriptor. The file descriptor is used by other I/O functions that refer to that pseudo-terminal.

28376 The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

28378 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in <fcntl.h>:

28380 O_RDWR Open for reading and writing.

28381 O_NOCTTY If set *posix_openpt()* shall not cause the terminal device to become the controlling terminal for the process.

28383 The behavior of other values for the *oflag* argument is unspecified.

28384 **RETURN VALUE**

28385 Upon successful completion, the *posix_openpt()* function shall open a master pseudo-terminal device and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, -1 shall be returned and *errno* set to indicate the error.

28388 **ERRORS**

28389 The *posix_openpt()* function shall fail if:

28390 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

28391 [ENFILE] The maximum allowable number of files is currently open in the system.

28392 The *posix_openpt()* function may fail if:

28393 [EINVAL] The value of *oflag* is not valid.

28394 [EAGAIN] Out of pseudo-terminal resources.

28395 xSR [ENOSR] Out of STREAMS resources.

28396 **EXAMPLES**

28397 **Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File Descriptor**

28399 #include <fcntl.h>

28400 #include <stdio.h>

28401 int masterfd, slavefd;

28402 char *slavedevice;

28403 masterfd = posix_openpt(O_RDWR|O_NOCTTY);

28404 if (masterfd == -1

28405 || grantpt(masterfd) == -1

```
28406         || unlockpt (masterfd) == -1
28407         || (slavedevice = ptsname (masterfd)) == NULL)
28408         return -1;

28409     printf("slave device is: %s\n", slavedevice);

28410     slavefd = open(slave, O_RDWR|O_NOCTTY);
28411     if (slavefd < 0)
28412         return -1;
```

28413 APPLICATION USAGE

28414 This function is a method for portably obtaining a file descriptor of a master terminal device for
28415 a pseudo-terminal. The *grantpt()* and *ptsname()* functions can be used to manipulate mode and
28416 ownership permissions, and to obtain the name of the slave device, respectively.

28417 RATIONALE

28418 The standard developers considered the matter of adding a special device for cloning master
28419 pseudo-terminals: the **/dev/ptmx** device. However, consensus could not be reached, and it was
28420 felt that adding a new function would permit other implementations. The *posix_openpt()*
28421 function is designed to complement the *grantpt()*, *ptsname()*, and *unlockpt()* functions.

28422 On implementations supporting the **/dev/ptmx** clone device, opening the master device of a
28423 pseudo-terminal is simply:

```
28424     mfdp = open("/dev/ptmx", oflag );
28425     if (mfdp < 0)
28426         return -1;
```

28427 FUTURE DIRECTIONS

28428 None.

28429 SEE ALSO

28430 *grantpt()*, *open()*, *ptsname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-2001,
28431 <fcntl.h>

28432 CHANGE HISTORY

28433 First released in Issue 6.

28434 **NAME**28435 posix_spawn, posix_spawnnp — spawn a process (**ADVANCED REALTIME**)28436 **SYNOPSIS**28437 SPN `#include <spawn.h>`

```

28438 int posix_spawn(pid_t *restrict pid, const char *restrict path,
28439               const posix_spawn_file_actions_t *file_actions,
28440               const posix_spawnattr_t *restrict attrp,
28441               char *const argv[restrict], char *const envp[restrict]);
28442 int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
28443               const posix_spawn_file_actions_t *file_actions,
28444               const posix_spawnattr_t *restrict attrp,
28445               char *const argv[restrict], char * const envp[restrict]);
28446

```

28447 **DESCRIPTION**

28448 The *posix_spawn()* and *posix_spawnnp()* functions shall create a new process (child process) from
 28449 the specified process image. The new process image shall be constructed from a regular
 28450 executable file called the new process image file.

28451 When a C program is executed as the result of this call, it shall be entered as a C-language
 28452 function call as follows:

```
28453 int main(int argc, char *argv[]);
```

28454 where *argc* is the argument count and *argv* is an array of character pointers to the arguments
 28455 themselves. In addition, the following variable:

```
28456 extern char **environ;
```

28457 shall be initialized as a pointer to an array of character pointers to the environment strings.

28458 The argument *argv* is an array of character pointers to null-terminated strings. The last member
 28459 of this array shall be a null pointer and is not counted in *argc*. These strings constitute the
 28460 argument list available to the new process image. The value in *argv*[0] should point to a filename
 28461 that is associated with the process image being started by the *posix_spawn()* or *posix_spawnnp()*
 28462 function.

28463 The argument *envp* is an array of character pointers to null-terminated strings. These strings
 28464 constitute the environment for the new process image. The environment array is terminated by a
 28465 null pointer.

28466 The number of bytes available for the child process' combined argument and environment lists
 28467 is {ARG_MAX}. The implementation shall specify in the system documentation (see the Base
 28468 Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance) whether any list
 28469 overhead, such as length words, null terminators, pointers, or alignment bytes, is included in
 28470 this total.

28471 The *path* argument to *posix_spawn()* is a pathname that identifies the new process image file to
 28472 execute.

28473 The *file* parameter to *posix_spawnnp()* shall be used to construct a pathname that identifies the
 28474 new process image file. If the *file* parameter contains a slash character, the *file* parameter shall be
 28475 used as the pathname for the new process image file. Otherwise, the path prefix for this file shall
 28476 be obtained by a search of the directories passed as the environment variable *PATH* (see the Base
 28477 Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables). If this
 28478 environment variable is not defined, the results of the search are implementation-defined.

28479 If *file_actions* is a null pointer, then file descriptors open in the calling process shall remain open
 28480 in the child process, except for those whose close-on-exec flag FD_CLOEXEC is set (see *fcntl()*).
 28481 For those file descriptors that remain open, all attributes of the corresponding open file
 28482 descriptions, including file locks (see *fcntl()*), shall remain unchanged.

28483 If *file_actions* is not NULL, then the file descriptors open in the child process shall be those open
 28484 in the calling process as modified by the spawn file actions object pointed to by *file_actions* and
 28485 the FD_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have
 28486 been processed. The effective order of processing the spawn file actions shall be:

- 28487 1. The set of open file descriptors for the child process shall initially be the same set as is
 28488 open for the calling process. All attributes of the corresponding open file descriptions,
 28489 including file locks (see *fcntl()*), shall remain unchanged.
- 28490 2. The signal mask, signal default actions, and the effective user and group IDs for the child
 28491 process shall be changed as specified in the attributes object referenced by *attrp*.
- 28492 3. The file actions specified by the spawn file actions object shall be performed in the order in
 28493 which they were added to the spawn file actions object.
- 28494 4. Any file descriptor that has its FD_CLOEXEC flag set (see *fcntl()*) shall be closed.

28495 The **posix_spawnattr_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at
 28496 least the attributes defined below.

28497 If the POSIX_SPAWN_SETPGROUP flag is set in the *spawn-flags* attribute of the object
 28498 referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the
 28499 child's process group shall be as specified in the *spawn-pgroup* attribute of the object referenced
 28500 by *attrp*.

28501 As a special case, if the POSIX_SPAWN_SETPGROUP flag is set in the *spawn-flags* attribute of
 28502 the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero,
 28503 then the child shall be in a new process group with a process group ID equal to its process ID.

28504 If the POSIX_SPAWN_SETPGROUP flag is not set in the *spawn-flags* attribute of the object
 28505 referenced by *attrp*, the new child process shall inherit the parent's process group.

28506 PS If the POSIX_SPAWN_SETSCHEDPARAM flag is set in the *spawn-flags* attribute of the object
 28507 referenced by *attrp*, but POSIX_SPAWN_SETSCHEDULER is not set, the new process image
 28508 shall initially have the scheduling policy of the calling process with the scheduling parameters
 28509 specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

28510 If the POSIX_SPAWN_SETSCHEDULER flag is set in the *spawn-flags* attribute of the object
 28511 referenced by *attrp* (regardless of the setting of the POSIX_SPAWN_SETSCHEDPARAM flag),
 28512 the new process image shall initially have the scheduling policy specified in the *spawn-*
 28513 *schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in
 28514 the *spawn-schedparam* attribute of the same object.

28515 The POSIX_SPAWN_RESETPIDS flag in the *spawn-flags* attribute of the object referenced by *attrp*
 28516 governs the effective user ID of the child process. If this flag is not set, the child process shall
 28517 inherit the parent process' effective user ID. If this flag is set, the child process' effective user ID
 28518 shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit of the new
 28519 process image file is set, the effective user ID of the child process shall become that file's owner
 28520 ID before the new process image begins execution.

28521 The POSIX_SPAWN_RESETPIDS flag in the *spawn-flags* attribute of the object referenced by *attrp*
 28522 also governs the effective group ID of the child process. If this flag is not set, the child process
 28523 shall inherit the parent process' effective group ID. If this flag is set, the child process' effective
 28524 group ID shall be reset to the parent's real group ID. In either case, if the set-group-ID mode bit

28525 of the new process image file is set, the effective group ID of the child process shall become that
 28526 file's group ID before the new process image begins execution.

28527 If the POSIX_SPAWN_SETSIGMASK flag is set in the *spawn_flags* attribute of the object
 28528 referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-*
 28529 *sigmask* attribute of the object referenced by *attrp*.

28530 If the POSIX_SPAWN_SETSIGDEF flag is set in the *spawn_flags* attribute of the object referenced
 28531 by *attrp*, the signals specified in the *spawn-sigdefault* attribute of the same object shall be set to
 28532 their default actions in the child process. Signals set to the default action in the parent process
 28533 shall be set to the default action in the child process.

28534 Signals set to be caught by the calling process shall be set to the default action in the child
 28535 process.

28536 Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be
 28537 ignored by the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag
 28538 being set in the *spawn_flags* attribute of the object referenced by *attrp* and the signals being
 28539 indicated in the *spawn-sigdefault* attribute of the object referenced by *attrp*.

28540 If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the
 28541 SIGCHLD signal is set to be ignored or to the default action in the child process, unless
 28542 otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn_flags*
 28543 attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the
 28544 *spawn-sigdefault* attribute of the object referenced by *attrp*.

28545 If the value of the *attrp* pointer is NULL, then the default values are used.

28546 All process attributes, other than those influenced by the attributes set in the object referenced
 28547 by *attrp* as specified above or by the file descriptor manipulations specified in *file_actions*, shall
 28548 appear in the new process image as though *fork()* had been called to create a child process and
 28549 then a member of the *exec* family of functions had been called by the child process to execute the
 28550 new process image.

28551 THR It is implementation-defined whether the fork handlers are run when *posix_spawn()* or
 28552 *posix_spawnnp()* is called.

28553 RETURN VALUE

28554 Upon successful completion, *posix_spawn()* and *posix_spawnnp()* shall return the process ID of the
 28555 child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and
 28556 shall return zero as the function return value. Otherwise, no child process shall be created, the
 28557 value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number
 28558 shall be returned as the function return value to indicate the error. If the *pid* argument is a null
 28559 pointer, the process ID of the child is not returned to the caller.

28560 ERRORS

28561 The *posix_spawn()* and *posix_spawnnp()* functions may fail if:

28562 [EINVAL] The value specified by *file_actions* or *attrp* is invalid.

28563 If this error occurs after the calling process successfully returns from the *posix_spawn()* or
 28564 *posix_spawnnp()* function, the child process may exit with exit status 127.

28565 If *posix_spawn()* or *posix_spawnnp()* fail for any of the reasons that would cause *fork()* or one of
 28566 the *exec* family of functions to fail, an error value shall be returned as described by *fork()* and
 28567 *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child
 28568 process shall exit with exit status 127).

28569 If `POSIX_SPAWN_SETPGROUP` is set in the *spawn-flags* attribute of the object referenced by
 28570 *attrp*, and *posix_spawn()* or *posix_spawnnp()* fails while changing the child's process group, an
 28571 error value shall be returned as described by *setpgid()* (or, if the error occurs after the calling
 28572 process successfully returns, the child process shall exit with exit status 127).

28573 PS If `POSIX_SPAWN_SETSCHEDPARAM` is set and `POSIX_SPAWN_SETSCHEDULER` is not set
 28574 in the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix_spawn()* or
 28575 *posix_spawnnp()* fails for any of the reasons that would cause *sched_setparam()* to fail, an error
 28576 value shall be returned as described by *sched_setparam()* (or, if the error occurs after the calling
 28577 process successfully returns, the child process shall exit with exit status 127).

28578 If `POSIX_SPAWN_SETSCHEDULER` is set in the *spawn-flags* attribute of the object referenced by
 28579 *attrp*, and if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause
 28580 *sched_setscheduler()* to fail, an error value shall be returned as described by *sched_setscheduler()*
 28581 (or, if the error occurs after the calling process successfully returns, the child process shall exit
 28582 with exit status 127).

28583 If the *file_actions* argument is not `NULL`, and specifies any *close*, *dup2*, or *open* actions to be
 28584 performed, and if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause
 28585 *close()*, *dup2()*, or *open()* to fail, an error value shall be returned as described by *close()*, *dup2()*,
 28586 and *open()*, respectively (or, if the error occurs after the calling process successfully returns, the
 28587 child process shall exit with exit status 127). An open file action may, by itself, result in any of
 28588 the errors described by *close()* or *dup2()*, in addition to those described by *open()*.

28589 EXAMPLES

28590 None.

28591 APPLICATION USAGE

28592 These functions are part of the Spawn option and need not be provided on all implementations.

28593 RATIONALE

28594 The *posix_spawn()* function and its close relation *posix_spawnnp()* have been introduced to
 28595 overcome the following perceived difficulties with *fork()*: the *fork()* function is difficult or
 28596 impossible to implement without swapping or dynamic address translation.

- 28597 • Swapping is generally too slow for a realtime environment.
- 28598 • Dynamic address translation is not available everywhere that POSIX might be useful.
- 28599 • Processes are too useful to simply option out of POSIX whenever it must run without
- 28600 address translation or other MMU services.

28601 Thus, POSIX needs process creation and file execution primitives that can be efficiently
 28602 implemented without address translation or other MMU services.

28603 The *posix_spawn()* function is implementable as a library routine, but both *posix_spawn()* and
 28604 *posix_spawnnp()* are designed as kernel operations. Also, although they may be an efficient
 28605 replacement for many *fork()/exec* pairs, their goal is to provide useful process creation
 28606 primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for
 28607 *fork()/exec*.

28608 This view of the role of *posix_spawn()* and *posix_spawnnp()* influenced the design of their API. It
 28609 does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified
 28610 operations of any sort are permitted between the creation of the child process and the execution
 28611 of the new process image; any attempt to reach that level would need to provide a programming
 28612 language as parameters. Instead, *posix_spawn()* and *posix_spawnnp()* are process creation
 28613 primitives like the *Start_Process* and *Start_Process_Search* Ada language bindings package
 28614 *POSIX_Process_Primitives* and also like those in many operating systems that are not UNIX

systems, but with some POSIX-specific additions.

To achieve its coverage goals, *posix_spawn()* and *posix_spawnnp()* have control of six types of inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and whether each signal ignored in the parent will remain ignored in the child, or be reset to its default action in the child.

Control of file descriptors is required to allow an independently written child process image to access data streams opened by and even generated or read by the parent process without being specifically coded to know which parent files and file descriptors are to be used. Control of the process group ID is required to control how the child process' job control relates to that of the parent.

Control of the signal mask and signal defaulting is sufficient to support the implementation of *system()*. Although support for *system()* is not explicitly one of the goals for *posix_spawn()* and *posix_spawnnp()*, it is covered under the "at least 50%" coverage goal.

The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec* family of functions should fully specify open file inheritance. The implementation need make no decisions regarding the set of open file descriptors when the child process image begins execution, those decisions having already been made by the caller and expressed as the set of open file descriptors and their *FD_CLOEXEC* flags at the time of the call and the spawn file actions object specified in the call. We have been assured that in cases where the POSIX *Start_Process* Ada primitives have been implemented in a library, this method of controlling file descriptor inheritance may be implemented very easily.

We can identify several problems with *posix_spawn()* and *posix_spawnnp()*, but there does not appear to be a solution that introduces fewer problems. Environment modification for child process attributes not specifiable via the *attrp* or *file_actions* arguments must be done in the parent process, and since the parent generally wants to save its context, it is more costly than similar functionality with *fork()/exec*. It is also complicated to modify the environment of a multi-threaded process temporarily, since all threads must agree when it is safe for the environment to be changed. However, this cost is only borne by those invocations of *posix_spawn()* and *posix_spawnnp()* that use the additional functionality. Since extensive modifications are not the usual case, and are particularly unlikely in time-critical code, keeping much of the environment control out of *posix_spawn()* and *posix_spawnnp()* is appropriate design.

The *posix_spawn()* and *posix_spawnnp()* functions do not have all the power of *fork()/exec*. This is to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to duplicate its functionality in a simple, fast function with no special hardware requirements. It is worth noting that *posix_spawn()* and *posix_spawnnp()* are very similar to the process creation operations on many operating systems that are not UNIX systems.

Requirements

The requirements for *posix_spawn()* and *posix_spawnnp()* are:

- They must be implementable without an MMU or unusual hardware.
- They must be compatible with existing POSIX standards.

Additional goals are:

- They should be efficiently implementable.
- They should be able to replace at least 50% of typical executions of *fork()*.

28659 • A system with *posix_spawn()* and *posix_spawnnp()* and without *fork()* should be useful, at least
28660 for realtime applications.

28661 • A system with *fork()* and the *exec* family should be able to implement *posix_spawn()* and
28662 *posix_spawnnp()* as library routines.

28663 Two-Syntax

28664 POSIX *exec* has several calling sequences with approximately the same functionality. These
28665 appear to be required for compatibility with existing practice. Since the existing practice for the
28666 *posix_spawn*()* functions is otherwise substantially unlike POSIX, we feel that simplicity
28667 outweighs compatibility. There are, therefore, only two names for the *posix_spawn*()* functions.

28668 The parameter list does not differ between *posix_spawn()* and *posix_spawnnp()*; *posix_spawnnp()*
28669 interprets the second parameter more elaborately than *posix_spawn()*.

28670 Compatibility with POSIX.5 (Ada)

28671 The *Start_Process* and *Start_Process_Search* procedures from the *POSIX_Process_Primitives*
28672 package from the Ada language binding to POSIX.1 encapsulate *fork()* and *exec* functionality in a
28673 manner similar to that of *posix_spawn()* and *posix_spawnnp()*. Originally, in keeping with our
28674 simplicity goal, the standard developers had limited the capabilities of *posix_spawn()* and
28675 *posix_spawnnp()* to a subset of the capabilities of *Start_Process* and *Start_Process_Search*; certain
28676 non-default capabilities were not supported. However, based on suggestions by the ballot group
28677 to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings
28678 working group member, the standard developers decided that *posix_spawn()* and *posix_spawnnp()*
28679 should be sufficiently powerful to implement *Start_Process* and *Start_Process_Search*. The
28680 rationale is that if the Ada language binding to such a primitive had already been approved as
28681 an IEEE standard, there can be little justification for not approving the functionally-equivalent
28682 parts of a C binding. The only three capabilities provided by *posix_spawn()* and *posix_spawnnp()*
28683 that are not provided by *Start_Process* and *Start_Process_Search* are optionally specifying the
28684 child's process group ID, the set of signals to be reset to default signal handling in the child
28685 process, and the child's scheduling policy and parameters.

28686 For the Ada language binding for *Start_Process* to be implemented with *posix_spawn()*, that
28687 binding would need to explicitly pass an empty signal mask and the parent's environment to
28688 *posix_spawn()* whenever the caller of *Start_Process* allowed these arguments to default, since
28689 *posix_spawn()* does not provide such defaults. The ability of *Start_Process* to mask user-specified
28690 signals during its execution is functionally unique to the Ada language binding and must be
28691 dealt with in the binding separately from the call to *posix_spawn()*.

28692 Process Group

28693 The process group inheritance field can be used to join the child process with an existing process
28694 group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by
28695 *attrp*, the *setpgid()* mechanism will place the child process in a new process group.

Threads

Without the *posix_spawn()* and *posix_spawnnp()* functions, systems without address translation can still use threads to give an abstraction of concurrency. In many cases, thread creation suffices, but it is not always a good substitute. The *posix_spawn()* and *posix_spawnnp()* functions are considerably “heavier” than thread creation. Processes have several important attributes that threads do not. Even without address translation, a process may have base-and-bound memory protection. Each process has a process environment including security attributes and file capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-uniform-memory-architecture multi-processors better than threads, and they are more convenient to use for activities that are not closely linked.

The *posix_spawn()* and *posix_spawnnp()* functions may not bring support for multiple processes to every configuration. Process creation is not the only piece of operating system support required to support multiple processes. The total cost of support for multiple processes may be quite high in some circumstances. Existing practice shows that support for multiple processes is uncommon and threads are common among “tiny kernels”. There should, therefore, probably continue to be AEPs for operating systems with only one process.

Asynchronous Error Notification

A library implementation of *posix_spawn()* or *posix_spawnnp()* may not be able to detect all possible errors before it forks the child process. IEEE Std 1003.1-2001 provides for an error indication returned from a child process which could not successfully complete the spawn operation via a special exit status which may be detected using the status value returned by *wait()* and *waitpid()*.

The *stat_val* interface and the macros used to interpret it are not well suited to the purpose of returning API errors, but they are the only path available to a library implementation. Thus, an implementation may cause the child process to exit with exit status 127 for any error detected during the spawn process after the *posix_spawn()* or *posix_spawnnp()* function has successfully returned.

The standard developers had proposed using two additional macros to interpret *stat_val*. The first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because of an error detected during the *posix_spawn()* or *posix_spawnnp()* operations rather than during actual execution of the child process image; the second, WSPAWNERRNO, would have extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group strongly opposed this because it would make a library implementation of *posix_spawn()* or *posix_spawnnp()* dependent on kernel modifications to *waitpid()* to be able to embed special information in *stat_val* to indicate a spawn failure.

The 8 bits of child process exit status that are guaranteed by IEEE Std 1003.1-2001 to be accessible to the waiting parent process are insufficient to disambiguate a spawn error from any other kind of error that may be returned by an arbitrary process image. No other bits of the exit status are required to be visible in *stat_val*, so these macros could not be strictly implemented at the library level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this value by *system()* and *popen()* to signal failures in these operations that occur after the function has returned but before a shell is able to execute. The exit status of 127 does not uniquely identify this class of error, nor does it provide any detailed information on the nature of the failure. Note that a kernel implementation of *posix_spawn()* or *posix_spawnnp()* is permitted (and encouraged) to return any possible error as the function value, thus providing more detailed failure information to the parent process.

Thus, no special macros are available to isolate asynchronous *posix_spawn()* or *posix_spawnnp()* errors. Instead, errors detected by the *posix_spawn()* or *posix_spawnnp()* operations in the context

28744 of the child process before the new process image executes are reported by setting the child's
 28745 exit status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on
 28746 the *stat_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that
 28747 other status values with which the child process image may exit (before the parent can
 28748 conclusively determine that the child process image has begun execution) are distinct from exit
 28749 status 127.

28750 FUTURE DIRECTIONS

28751 None.

28752 SEE ALSO

28753 *alarm()*, *chmod()*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *kill()*, *open()*,
 28754 *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*,
 28755 *posix_spawn_file_actions_addopen()*, *posix_spawn_file_actions_destroy()*, <REFERENCE
 28756 UNDEFINED>(*posix_spawn_file_actions_init()*, *posix_spawnattr_destroy()*, *posix_spawnattr_init()*,
 28757 *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*,
 28758 *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*,
 28759 *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*, *posix_spawnattr_setpgroup()*,
 28760 *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*,
 28761 *sched_setparam()*, *sched_setscheduler()*, *setpgid()*, *setuid()*, *stat()*, *times()*, *wait()*, the Base
 28762 Definitions volume of IEEE Std 1003.1-2001, <**spawn.h**>

28763 CHANGE HISTORY

28764 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28765 IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are
 28766 changed as well as the signal mask in step 2.

28767 IEEE PASC Interpretation 1003.1 #132 is applied.

28768 NAME

28769 `posix_spawn_file_actions_addclose`, `posix_spawn_file_actions_addopen` — add close or open
 28770 action to spawn file actions object (**ADVANCED REALTIME**)

28771 SYNOPSIS

```
28772 SPN      #include <spawn.h>

28773      int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *
28774          file_actions, int fildes);
28775      int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *
28776          restrict file_actions, int fildes,
28777          const char *restrict path, int oflag, mode_t mode);
28778
```

28779 DESCRIPTION

28780 These functions shall add or delete a close or open action to a spawn file actions object.

28781 A spawn file actions object is of type **posix_spawn_file_actions_t** (defined in `<spawn.h>`) and is
 28782 used to specify a series of actions to be performed by a `posix_spawn()` or `posix_spawnnp()`
 28783 operation in order to arrive at the set of open file descriptors for the child process given the set of
 28784 open file descriptors of the parent. IEEE Std 1003.1-2001 does not define comparison or
 28785 assignment operators for the type **posix_spawn_file_actions_t**.

28786 A spawn file actions object, when passed to `posix_spawn()` or `posix_spawnnp()`, shall specify how
 28787 the set of open file descriptors in the calling process is transformed into a set of potentially open
 28788 file descriptors for the spawned process. This transformation shall be as if the specified sequence
 28789 of actions was performed exactly once, in the context of the spawned process (prior to execution
 28790 of the new process image), in the order in which the actions were added to the object;
 28791 additionally, when the new process image is executed, any file descriptor (from this new set)
 28792 which has its `FD_CLOEXEC` flag set shall be closed (see `posix_spawn()`).

28793 The `posix_spawn_file_actions_addclose()` function shall add a *close* action to the object referenced
 28794 by *file_actions* that shall cause the file descriptor *fildes* to be closed (as if `close(fildes)` had been
 28795 called) when a new process is spawned using this file actions object.

28796 The `posix_spawn_file_actions_addopen()` function shall add an *open* action to the object referenced
 28797 by *file_actions* that shall cause the file named by *path* to be opened (as if `open(path, oflag, mode)`
 28798 had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a
 28799 new process is spawned using this file actions object. If *fildes* was already an open file descriptor,
 28800 it shall be closed before the new file is opened.

28801 The string described by *path* shall be copied by the `posix_spawn_file_actions_addopen()` function.

28802 RETURN VALUE

28803 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 28804 be returned to indicate the error.

28805 ERRORS

28806 These functions shall fail if:

28807 [EBADF] The value specified by *fildes* is negative or greater than or equal to
 28808 {OPEN_MAX}.

28809 These functions may fail if:

28810 [EINVAL] The value specified by *file_actions* is invalid.

28811 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

28812 It shall not be considered an error for the *files* argument passed to these functions to specify a
 28813 file descriptor for which the specified operation could not be performed at the time of the call.
 28814 Any such error will be detected when the associated file actions object is later used during a
 28815 *posix_spawn()* or *posix_spawnnp()* operation.

28816 EXAMPLES

28817 None.

28818 APPLICATION USAGE

28819 These functions are part of the Spawn option and need not be provided on all implementations.

28820 RATIONALE

28821 A spawn file actions object may be initialized to contain an ordered sequence of *close()*, *dup2()*,
 28822 and *open()* operations to be used by *posix_spawn()* or *posix_spawnnp()* to arrive at the set of open
 28823 file descriptors inherited by the spawned process from the set of open file descriptors in the
 28824 parent at the time of the *posix_spawn()* or *posix_spawnnp()* call. It had been suggested that the
 28825 *close()* and *dup2()* operations alone are sufficient to rearrange file descriptors, and that files
 28826 which need to be opened for use by the spawned process can be handled either by having the
 28827 calling process open them before the *posix_spawn()* or *posix_spawnnp()* call (and close them after),
 28828 or by passing filenames to the spawned process (in *argv*) so that it may open them itself. The
 28829 standard developers recommend that applications use one of these two methods when practical,
 28830 since detailed error status on a failed open operation is always available to the application this
 28831 way. However, the standard developers feel that allowing a spawn file actions object to specify
 28832 open operations is still appropriate because:

- 28833 1. It is consistent with equivalent POSIX.5 (Ada) functionality.
- 28834 2. It supports the I/O redirection paradigm commonly employed by POSIX programs
 28835 designed to be invoked from a shell. When such a program is the child process, it may not
 28836 be designed to open files on its own.
- 28837 3. It allows file opens that might otherwise fail or violate file ownership/access rights if
 28838 executed by the parent process.

28839 Regarding 2. above, note that the spawn open file action provides to *posix_spawn()* and
 28840 *posix_spawnnp()* the same capability that the shell redirection operators provide to *system()*, only
 28841 without the intervening execution of a shell; for example:

```
28842 system ("myprog <file1 3<file2");
```

28843 Regarding 3. above, note that if the calling process needs to open one or more files for access by
 28844 the spawned process, but has insufficient spare file descriptors, then the open action is necessary
 28845 to allow the *open()* to occur in the context of the child process after other file descriptors have
 28846 been closed (that must remain open in the parent).

28847 Additionally, if a parent is executed from a file having a “set-user-id” mode bit set and the
 28848 POSIX_SPAWN_RESETPIDS flag is set in the spawn attributes, a file created within the parent
 28849 process will (possibly incorrectly) have the parent’s effective user ID as its owner, whereas a file
 28850 created via an *open()* action during *posix_spawn()* or *posix_spawnnp()* will have the parent’s real
 28851 ID as its owner; and an open by the parent process may successfully open a file to which the real
 28852 user should not have access or fail to open a file to which the real user should have access.

28853 File Descriptor Mapping

28854 The standard developers had originally proposed using an array which specified the mapping of
28855 child file descriptors back to those of the parent. It was pointed out by the ballot group that it is
28856 not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix_spawn()*
28857 or *posix_spawnnp()* without provision for one or more spare file descriptor entries (which simply
28858 may not be available). Such an array requires that an implementation develop a complex
28859 strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor
28860 at the wrong time.

28861 It was noted by a member of the Ada Language Bindings working group that the approved Ada
28862 Language *Start_Process* family of POSIX process primitives use a caller-specified set of file
28863 actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very
28864 flexible way, yet no such problems exist because the burden of determining how to achieve the
28865 final file descriptor mapping is completely on the application. Furthermore, although the file
28866 actions interface appears frightening at first glance, it is actually quite simple to implement in
28867 either a library or the kernel.

28868 FUTURE DIRECTIONS

28869 None.

28870 SEE ALSO

28871 *close()*, *dup()*, *open()*, *posix_spawn()*, *posix_spawn_file_actions_adddup2()*,
28872 *posix_spawn_file_actions_destroy()*, *posix_spawnnp()*, the Base Definitions volume of
28873 IEEE Std 1003.1-2001, <**spawn.h**>

28874 CHANGE HISTORY

28875 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28876 IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the
28877 string pointed to by *path* is copied by the *posix_spawn_file_actions_addopen()* function.

28878 NAME

28879 `posix_spawn_file_actions_adddup2` — add `dup2` action to spawn file actions object
28880 (**ADVANCED REALTIME**)

28881 SYNOPSIS

28882 SPN `#include <spawn.h>`

```
28883        int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *
28884                                                file_actions, int fildes, int newfildes);
```

28885

28886 DESCRIPTION

28887 The `posix_spawn_file_actions_adddup2()` function shall add a `dup2()` action to the object
28888 referenced by `file_actions` that shall cause the file descriptor `fildes` to be duplicated as `newfildes` (as
28889 if `dup2(fildes, newfildes)` had been called) when a new process is spawned using this file actions
28890 object.

28891 A spawn file actions object is as defined in `posix_spawn_file_actions_addclose()`.

28892 RETURN VALUE

28893 Upon successful completion, the `posix_spawn_file_actions_adddup2()` function shall return zero;
28894 otherwise, an error number shall be returned to indicate the error.

28895 ERRORS

28896 The `posix_spawn_file_actions_adddup2()` function shall fail if:

28897 [EBADF] The value specified by `fildes` or `newfildes` is negative or greater than or equal to
28898 {OPEN_MAX}.

28899 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

28900 The `posix_spawn_file_actions_adddup2()` function may fail if:

28901 [EINVAL] The value specified by `file_actions` is invalid.

28902 It shall not be considered an error for the `fildes` argument passed to the
28903 `posix_spawn_file_actions_adddup2()` function to specify a file descriptor for which the specified
28904 operation could not be performed at the time of the call. Any such error will be detected when
28905 the associated file actions object is later used during a `posix_spawn()` or `posix_spawnnp()`
28906 operation.

28907 EXAMPLES

28908 None.

28909 APPLICATION USAGE

28910 The `posix_spawn_file_actions_adddup2()` function is part of the Spawn option and need not be
28911 provided on all implementations.

28912 RATIONALE

28913 Refer to the RATIONALE in `posix_spawn_file_actions_addclose()`.

28914 FUTURE DIRECTIONS

28915 None.

28916 SEE ALSO

28917 `dup()`, `posix_spawn()`, `posix_spawn_file_actions_addclose()`, `posix_spawn_file_actions_destroy()`,
28918 `posix_spawnnp()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

CHANGE HISTORY

- 28919 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
- 28920
- 28921 IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the
- 28922 *newfildes* argument in addition to *fildes*.

28923 NAME

28924 posix_spawn_file_actions_addopen — add open action to spawn file actions object
28925 **(ADVANCED REALTIME)**

28926 SYNOPSIS

```
28927 SPN   #include <spawn.h>  
  
28928       int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *  
28929       restrict file_actions, int fildes,  
28930       const char *restrict path, int oflag, mode_t mode);  
28931
```

28932 DESCRIPTION

28933 Refer to *posix_spawn_file_actions_addclose()*.

28934 NAME

28935 `posix_spawn_file_actions_destroy`, `posix_spawn_file_actions_init` — destroy and initialize
 28936 spawn file actions object (**ADVANCED REALTIME**)

28937 SYNOPSIS

28938 SPN `#include <spawn.h>`

```
28939 int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *
28940     file_actions);
28941 int posix_spawn_file_actions_init(posix_spawn_file_actions_t *
28942     file_actions);
28943
```

28944 DESCRIPTION

28945 The `posix_spawn_file_actions_destroy()` function shall destroy the object referenced by `file_actions`;
 28946 the object becomes, in effect, uninitialized. An implementation may cause
 28947 `posix_spawn_file_actions_destroy()` to set the object referenced by `file_actions` to an invalid value. A
 28948 destroyed spawn file actions object can be reinitialized using `posix_spawn_file_actions_init()`; the
 28949 results of otherwise referencing the object after it has been destroyed are undefined.

28950 The `posix_spawn_file_actions_init()` function shall initialize the object referenced by `file_actions` to
 28951 contain no file actions for `posix_spawn()` or `posix_spawnnp()` to perform.

28952 A spawn file actions object is as defined in `posix_spawn_file_actions_addclose()`.

28953 The effect of initializing an already initialized spawn file actions object is undefined.

28954 RETURN VALUE

28955 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 28956 be returned to indicate the error.

28957 ERRORS

28958 The `posix_spawn_file_actions_init()` function shall fail if:

28959 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

28960 The `posix_spawn_file_actions_destroy()` function may fail if:

28961 [EINVAL] The value specified by `file_actions` is invalid.

28962 EXAMPLES

28963 None.

28964 APPLICATION USAGE

28965 These functions are part of the Spawn option and need not be provided on all implementations.

28966 RATIONALE

28967 Refer to the RATIONALE in `posix_spawn_file_actions_addclose()`.

28968 FUTURE DIRECTIONS

28969 None.

28970 SEE ALSO

28971 `posix_spawn()`, `posix_spawnnp()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

28972 CHANGE HISTORY

28973 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28974 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

28975 NAME

28976 `posix_spawnattr_destroy`, `posix_spawnattr_init` — destroy and initialize spawn attributes object
28977 (**ADVANCED REALTIME**)

28978 SYNOPSIS

28979 SPN `#include <spawn.h>`

28980 `int posix_spawnattr_destroy(posix_spawnattr_t *attr);`

28981 `int posix_spawnattr_init(posix_spawnattr_t *attr);`

28982

28983 DESCRIPTION

28984 The `posix_spawnattr_destroy()` function shall destroy a spawn attributes object. A destroyed *attr*
28985 attributes object can be reinitialized using `posix_spawnattr_init()`; the results of otherwise
28986 referencing the object after it has been destroyed are undefined. An implementation may cause
28987 `posix_spawnattr_destroy()` to set the object referenced by *attr* to an invalid value.

28988 The `posix_spawnattr_init()` function shall initialize a spawn attributes object *attr* with the default
28989 value for all of the individual attributes used by the implementation. Results are undefined if
28990 `posix_spawnattr_init()` is called specifying an already initialized *attr* attributes object.

28991 A spawn attributes object is of type **posix_spawnattr_t** (defined in **<spawn.h>**) and is used to
28992 specify the inheritance of process attributes across a spawn operation. IEEE Std 1003.1-2001 does
28993 not define comparison or assignment operators for the type **posix_spawnattr_t**.

28994 Each implementation shall document the individual attributes it uses and their default values
28995 unless these values are defined by IEEE Std 1003.1-2001. Attributes not defined by
28996 IEEE Std 1003.1-2001, their default values, and the names of the associated functions to get and
28997 set those attribute values are implementation-defined.

28998 The resulting spawn attributes object (possibly modified by setting individual attribute values),
28999 is used to modify the behavior of `posix_spawn()` or `posix_spawnnp()`. After a spawn attributes
29000 object has been used to spawn a process by a call to a `posix_spawn()` or `posix_spawnnp()`, any
29001 function affecting the attributes object (including destruction) shall not affect any process that
29002 has been spawned in this way.

29003 RETURN VALUE

29004 Upon successful completion, `posix_spawnattr_destroy()` and `posix_spawnattr_init()` shall return
29005 zero; otherwise, an error number shall be returned to indicate the error.

29006 ERRORS

29007 The `posix_spawnattr_init()` function shall fail if:

29008 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

29009 The `posix_spawnattr_destroy()` function may fail if:

29010 [EINVAL] The value specified by *attr* is invalid.

29011 EXAMPLES

29012 None.

29013 APPLICATION USAGE

29014 These functions are part of the Spawn option and need not be provided on all implementations.

29015 RATIONALE

29016 The original spawn interface proposed in IEEE Std 1003.1-2001 defined the attributes that specify
29017 the inheritance of process attributes across a spawn operation as a structure. In order to be able
29018 to separate optional individual attributes under their appropriate options (that is, the *spawn-*
29019 *schedparam* and *spawn-schedpolicy* attributes depending upon the Process Scheduling option), and

also for extensibility and consistency with the newer POSIX interfaces, the attributes interface has been changed to an opaque data type. This interface now consists of the type **posix_spawnattr_t**, representing a spawn attributes object, together with associated functions to initialize or destroy the attributes object, and to set or get each individual attribute. Although the new object-oriented interface is more verbose than the original structure, it is simple to use, more extensible, and easy to implement.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn(), *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*,
posix_spawnattr_getpgroup(), *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
posix_spawnattr_getsigmask(), *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*,
posix_spawnattr_setpgroup(), *posix_spawnattr_setsigmask()*, *posix_spawnattr_setschedpolicy()*,
posix_spawnattr_setschedparam(), *posix_spawnnp()*, the Base Definitions volume of
IEEE Std 1003.1-2001, <**spawn.h**>

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already initialized spawn attributes option is undefined.

29039 NAME

29040 posix_spawnattr_getflags, posix_spawnattr_setflags — get and set the spawn-flags attribute of a
29041 spawn attributes object (**ADVANCED REALTIME**)

29042 SYNOPSIS

```
29043 SPN    #include <spawn.h>

29044    int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
29045                                short *restrict flags);
29046    int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
29047
```

29048 DESCRIPTION

29049 The *posix_spawnattr_getflags()* function shall obtain the value of the *spawn-flags* attribute from
29050 the attributes object referenced by *attr*.

29051 The *posix_spawnattr_setflags()* function shall set the *spawn-flags* attribute in an initialized
29052 attributes object referenced by *attr*.

29053 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the
29054 new process image when invoking *posix_spawn()* or *posix_spawnp()*. It is the bitwise-inclusive
29055 OR of zero or more of the following flags:

```
29056 POSIX_SPAWN_RESETEIDS
29057 POSIX_SPAWN_SETPGROUP
29058 POSIX_SPAWN_SETSIGDEF
29059 POSIX_SPAWN_SETSIGMASK
29060 PS  POSIX_SPAWN_SETSCHEDPARAM
29061    POSIX_SPAWN_SETSCHEDULER
29062
```

29063 These flags are defined in **<spawn.h>**. The default value of this attribute shall be as if no flags
29064 were set.

29065 RETURN VALUE

29066 Upon successful completion, *posix_spawnattr_getflags()* shall return zero and store the value of
29067 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an
29068 error number shall be returned to indicate the error.

29069 Upon successful completion, *posix_spawnattr_setflags()* shall return zero; otherwise, an error
29070 number shall be returned to indicate the error.

29071 ERRORS

29072 These functions may fail if:

29073 [EINVAL] The value specified by *attr* is invalid.

29074 The *posix_spawnattr_setflags()* function may fail if:

29075 [EINVAL] The value of the attribute being set is not valid.

29076 EXAMPLES

29077 None.

29078 APPLICATION USAGE

29079 These functions are part of the Spawn option and need not be provided on all implementations.

29080 RATIONALE

29081 None.

29082 FUTURE DIRECTIONS

29083 None.

29084 SEE ALSO

29085 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_init()*, *posix_spawnattr_getsigdefault()*,
29086 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
29087 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setpgroup()*,
29088 *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*,
29089 *posix_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**spawn.h**>

29090 CHANGE HISTORY

29091 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29092 NAME

29093 posix_spawnattr_getpgroup, posix_spawnattr_setpgroup — get and set the spawn-pgroup
29094 attribute of a spawn attributes object (**ADVANCED REALTIME**)

29095 SYNOPSIS

```
29096 SPN    #include <spawn.h>

29097    int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
29098                                pid_t *restrict pgroup);
29099    int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
29100
```

29101 DESCRIPTION

29102 The *posix_spawnattr_getpgroup()* function shall obtain the value of the *spawn-pgroup* attribute
29103 from the attributes object referenced by *attr*.

29104 The *posix_spawnattr_setpgroup()* function shall set the *spawn-pgroup* attribute in an initialized
29105 attributes object referenced by *attr*.

29106 The *spawn-pgroup* attribute represents the process group to be joined by the new process image
29107 in a spawn operation (if POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute). The
29108 default value of this attribute shall be zero.

29109 RETURN VALUE

29110 Upon successful completion, *posix_spawnattr_getpgroup()* shall return zero and store the value of
29111 the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise,
29112 an error number shall be returned to indicate the error.

29113 Upon successful completion, *posix_spawnattr_setpgroup()* shall return zero; otherwise, an error
29114 number shall be returned to indicate the error.

29115 ERRORS

29116 These functions may fail if:

29117 [EINVAL] The value specified by *attr* is invalid.

29118 The *posix_spawnattr_setpgroup()* function may fail if:

29119 [EINVAL] The value of the attribute being set is not valid.

29120 EXAMPLES

29121 None.

29122 APPLICATION USAGE

29123 These functions are part of the Spawn option and need not be provided on all implementations.

29124 RATIONALE

29125 None.

29126 FUTURE DIRECTIONS

29127 None.

29128 SEE ALSO

29129 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_init()*, *posix_spawnattr_getsigdefault()*,
29130 *posix_spawnattr_getflags()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
29131 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*,
29132 *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*,
29133 *posix_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <spawn.h>

29134 **CHANGE HISTORY**

29135 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29136 NAME

29137 posix_spawnattr_getschedparam, posix_spawnattr_setschedparam — get and set the spawn-
29138 schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

29139 SYNOPSIS

29140 SPN PS #include <spawn.h>

29141 #include <sched.h>

```
29142 int posix_spawnattr_getschedparam(const posix_spawnattr_t *
29143     restrict attr, struct sched_param *restrict schedparam);
29144 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
29145     const struct sched_param *restrict schedparam);
```

29146

29147 DESCRIPTION

29148 The *posix_spawnattr_getschedparam()* function shall obtain the value of the *spawn-schedparam*
29149 attribute from the attributes object referenced by *attr*.

29150 The *posix_spawnattr_setschedparam()* function shall set the *spawn-schedparam* attribute in an
29151 initialized attributes object referenced by *attr*.

29152 The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new
29153 process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER or
29154 POSIX_SPAWN_SETSCHEDPARAM is set in the *spawn-flags* attribute). The default value of this
29155 attribute is unspecified.

29156 RETURN VALUE

29157 Upon successful completion, *posix_spawnattr_getschedparam()* shall return zero and store the
29158 value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam*
29159 parameter; otherwise, an error number shall be returned to indicate the error.

29160 Upon successful completion, *posix_spawnattr_setschedparam()* shall return zero; otherwise, an
29161 error number shall be returned to indicate the error.

29162 ERRORS

29163 These functions may fail if:

29164 [EINVAL] The value specified by *attr* is invalid.

29165 The *posix_spawnattr_setschedparam()* function may fail if:

29166 [EINVAL] The value of the attribute being set is not valid.

29167 EXAMPLES

29168 None.

29169 APPLICATION USAGE

29170 These functions are part of the Spawn and Process Scheduling options and need not be provided
29171 on all implementations.

29172 RATIONALE

29173 None.

29174 FUTURE DIRECTIONS

29175 None.

29176 SEE ALSO

29177 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_init()*, *posix_spawnattr_getsigdefault()*,
29178 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedpolicy()*,
29179 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*,

29180 *posix_spawnattr_setpgroup()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*,
29181 *posix_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sched.h>**, **<spawn.h>**

29182 **CHANGE HISTORY**

29183 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29184 NAME

29185 posix_spawnattr_getschedpolicy, posix_spawnattr_setschedpolicy — get and set the spawn-
29186 schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

29187 SYNOPSIS

29188 SPN PS #include <spawn.h>

29189 #include <sched.h>

```
29190 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *
29191     restrict attr, int *restrict schedpolicy);
29192 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
29193     int schedpolicy);
29194
```

29195 DESCRIPTION

29196 The *posix_spawnattr_getschedpolicy()* function shall obtain the value of the *spawn-schedpolicy*
29197 attribute from the attributes object referenced by *attr*.

29198 The *posix_spawnattr_setschedpolicy()* function shall set the *spawn-schedpolicy* attribute in an
29199 initialized attributes object referenced by *attr*.

29200 The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new
29201 process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER is set in the *spawn-*
29202 *flags* attribute). The default value of this attribute is unspecified.

29203 RETURN VALUE

29204 Upon successful completion, *posix_spawnattr_getschedpolicy()* shall return zero and store the
29205 value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy*
29206 parameter; otherwise, an error number shall be returned to indicate the error.

29207 Upon successful completion, *posix_spawnattr_setschedpolicy()* shall return zero; otherwise, an
29208 error number shall be returned to indicate the error.

29209 ERRORS

29210 These functions may fail if:

29211 [EINVAL] The value specified by *attr* is invalid.

29212 The *posix_spawnattr_setschedpolicy()* function may fail if:

29213 [EINVAL] The value of the attribute being set is not valid.

29214 EXAMPLES

29215 None.

29216 APPLICATION USAGE

29217 These functions are part of the Spawn and Process Scheduling options and need not be provided
29218 on all implementations.

29219 RATIONALE

29220 None.

29221 FUTURE DIRECTIONS

29222 None.

29223 SEE ALSO

29224 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_init()*, *posix_spawnattr_getsigdefault()*,
29225 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
29226 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*,
29227 *posix_spawnattr_setpgroup()*, *posix_spawnattr_setschedparam()*, *posix_spawnattr_setsigmask()*,

29228 *posix_spawnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sched.h**>, <**spawn.h**>

29229 **CHANGE HISTORY**

29230 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29231 NAME

29232 posix_spawnattr_getsigdefault, posix_spawnattr_setsigdefault — get and set the spawn-
29233 sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

29234 SYNOPSIS

```
29235 SPN    #include <signal.h>
29236        #include <spawn.h>

29237        int posix_spawnattr_getsigdefault(const posix_spawnattr_t *
29238            restrict attr, sigset_t *restrict sigdefault);
29239        int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
29240            const sigset_t *restrict sigdefault);
```

29242 DESCRIPTION

29243 The *posix_spawnattr_getsigdefault()* function shall obtain the value of the *spawn-sigdefault*
29244 attribute from the attributes object referenced by *attr*.

29245 The *posix_spawnattr_setsigdefault()* function shall set the *spawn-sigdefault* attribute in an
29246 initialized attributes object referenced by *attr*.

29247 The *spawn-sigdefault* attribute represents the set of signals to be forced to default signal handling
29248 in the new process image (if POSIX_SPAWN_SETSIGDEF is set in the *spawn-flags* attribute) by a
29249 spawn operation. The default value of this attribute shall be an empty signal set.

29250 RETURN VALUE

29251 Upon successful completion, *posix_spawnattr_getsigdefault()* shall return zero and store the value
29252 of the *spawn-sigdefault* attribute of *attr* into the object referenced by the *sigdefault* parameter;
29253 otherwise, an error number shall be returned to indicate the error.

29254 Upon successful completion, *posix_spawnattr_setsigdefault()* shall return zero; otherwise, an error
29255 number shall be returned to indicate the error.

29256 ERRORS

29257 These functions may fail if:

29258 [EINVAL] The value specified by *attr* is invalid.

29259 The *posix_spawnattr_setsigdefault()* function may fail if:

29260 [EINVAL] The value of the attribute being set is not valid.

29261 EXAMPLES

29262 None.

29263 APPLICATION USAGE

29264 These functions are part of the Spawn option and need not be provided on all implementations.

29265 RATIONALE

29266 None.

29267 FUTURE DIRECTIONS

29268 None.

29269 SEE ALSO

29270 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_init()*, *posix_spawnattr_getflags()*,
29271 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
29272 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setflags()*, *posix_spawnattr_setpgroup()*,
29273 *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*,
29274 *posix_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<spawn.h>**

CHANGE HISTORY

29276 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29277 NAME

29278 posix_spawnattr_getsigmask, posix_spawnattr_setsigmask — get and set the spawn-sigmask
29279 attribute of a spawn attributes object (**ADVANCED REALTIME**)

29280 SYNOPSIS

```
29281 SPN    #include <signal.h>
29282         #include <spawn.h>

29283         int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
29284         sigset_t *restrict sigmask);
29285         int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
29286         const sigset_t *restrict sigmask);
29287
```

29288 DESCRIPTION

29289 The *posix_spawnattr_getsigmask()* function shall obtain the value of the *spawn-sigmask* attribute
29290 from the attributes object referenced by *attr*.

29291 The *posix_spawnattr_setsigmask()* function shall set the *spawn-sigmask* attribute in an initialized
29292 attributes object referenced by *attr*.

29293 The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a
29294 spawn operation (if *POSIX_SPAWN_SETSIGMASK* is set in the *spawn-flags* attribute). The
29295 default value of this attribute is unspecified.

29296 RETURN VALUE

29297 Upon successful completion, *posix_spawnattr_getsigmask()* shall return zero and store the value
29298 of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter;
29299 otherwise, an error number shall be returned to indicate the error.

29300 Upon successful completion, *posix_spawnattr_setsigmask()* shall return zero; otherwise, an error
29301 number shall be returned to indicate the error.

29302 ERRORS

29303 These functions may fail if:

29304 [EINVAL] The value specified by *attr* is invalid.

29305 The *posix_spawnattr_setsigmask()* function may fail if:

29306 [EINVAL] The value of the attribute being set is not valid.

29307 EXAMPLES

29308 None.

29309 APPLICATION USAGE

29310 These functions are part of the Spawn option and need not be provided on all implementations.

29311 RATIONALE

29312 None.

29313 FUTURE DIRECTIONS

29314 None.

29315 SEE ALSO

29316 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_init()*, *posix_spawnattr_getsigdefault()*,
29317 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
29318 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*,
29319 *posix_spawnattr_setpgroup()*, *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*,
29320 *posix_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**, **<spawn.h>**

29321 **CHANGE HISTORY**

29322 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29323 NAME

29324 posix_spawnattr_init — initialize the spawn attributes object (**ADVANCED REALTIME**)

29325 SYNOPSIS

29326 SPN `#include <spawn.h>`

29327 `int posix_spawnattr_init(posix_spawnattr_t *attr);`

29328

29329 DESCRIPTION

29330 Refer to *posix_spawnattr_destroy()*.

29331 **NAME**

29332 posix_spawnattr_setflags — set the spawn-flags attribute of a spawn attributes object
29333 **(ADVANCED REALTIME)**

29334 **SYNOPSIS**

29335 SPN #include <spawn.h>

29336 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);

29337

29338 **DESCRIPTION**

29339 Refer to *posix_spawnattr_getflags()*.

29340 NAME

29341 posix_spawnattr_setpgroup — set the spawn-pgroup attribute of a spawn attributes object
29342 (**ADVANCED REALTIME**)

29343 SYNOPSIS

29344 SPN `#include <spawn.h>`

29345 `int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);`
29346

29347 DESCRIPTION

29348 Refer to *posix_spawnattr_getpgroup()*.

29349 **NAME**

29350 `posix_spawnattr_setschedparam` — set the spawn-schedparam attribute of a spawn attributes
29351 object (**ADVANCED REALTIME**)

29352 **SYNOPSIS**

29353 SPN PS `#include <sched.h>`

29354 `#include <spawn.h>`

```
29355        int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,  
29356                                          const struct sched_param *restrict schedparam);
```

29357

29358 **DESCRIPTION**

29359 Refer to `posix_spawnattr_getschedparam()`.

29360 NAME

29361 posix_spawnattr_setschedpolicy — set the spawn-schedpolicy attribute of a spawn attributes
29362 object (**ADVANCED REALTIME**)

29363 SYNOPSIS

29364 SPN PS #include <sched.h>

29365 #include <spawn.h>

```
29366        int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,  
29367                                          int schedpolicy);
```

29368

29369 DESCRIPTION

29370 Refer to *posix_spawnattr_getschedpolicy()*.

29371 **NAME**

29372 posix_spawnattr_setsigdefault — set the spawn-sigdefault attribute of a spawn attributes object
29373 **(ADVANCED REALTIME)**

29374 **SYNOPSIS**

29375 SPN #include <signal.h>

29376 #include <spawn.h>

29377 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
29378 const sigset_t *restrict sigdefault);

29379

29380 **DESCRIPTION**

29381 Refer to *posix_spawnattr_getsigdefault()*.

29382 NAME

29383 posix_spawnattr_setsigmask — set the spawn-sigmask attribute of a spawn attributes object
29384 **(ADVANCED REALTIME)**

29385 SYNOPSIS

```
29386 SPN     #include <signal.h>  
29387     #include <spawn.h>  
  
29388     int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,  
29389         const sigset_t *restrict sigmask);  
29390
```

29391 DESCRIPTION

29392 Refer to *posix_spawnattr_getsigmask()*.

29393 **NAME**29394 `posix_spawn` — spawn a process (**ADVANCED REALTIME**)29395 **SYNOPSIS**29396 SPN `#include <spawn.h>`

```
29397        int posix_spawn(pid_t *restrict pid, const char *restrict file,  
29398            const posix_spawn_file_actions_t *file_actions,  
29399            const posix_spawnattr_t *restrict attrp,  
29400            char *const argv[restrict], char *const envp[restrict]);  
29401
```

29402 **DESCRIPTION**29403 Refer to `posix_spawn()`.

29404 NAME

29405 `posix_trace_attr_destroy`, `posix_trace_attr_init` — destroy and initialize the trace stream
29406 attributes object (**TRACING**)

29407 SYNOPSIS

29408 TRC `#include <trace.h>`

29409 `int posix_trace_attr_destroy(trace_attr_t *attr);`

29410 `int posix_trace_attr_init(trace_attr_t *attr);`

29411

29412 DESCRIPTION

29413 The `posix_trace_attr_destroy()` function shall destroy an initialized trace attributes object. A
29414 destroyed `attr` attributes object can be reinitialized using `posix_trace_attr_init()`; the results of
29415 otherwise referencing the object after it has been destroyed are undefined.

29416 The `posix_trace_attr_init()` function shall initialize a trace attributes object `attr` with the default
29417 value for all of the individual attributes used by a given implementation. The read-only
29418 *generation-version* and *clock-resolution* attributes of the newly initialized trace attributes object
29419 shall be set to their appropriate values (see Section 2.11.1.2 (on page 75)).

29420 Results are undefined if `posix_trace_attr_init()` is called specifying an already initialized `attr`
29421 attributes object.

29422 Implementations may add extensions to the trace attributes object structure as permitted in the
29423 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance.

29424 The resulting attributes object (possibly modified by setting individual attributes values), when
29425 used by `posix_trace_create()`, defines the attributes of the trace stream created. A single attributes
29426 object can be used in multiple calls to `posix_trace_create()`. After one or more trace streams have
29427 been created using an attributes object, any function affecting that attributes object, including
29428 destruction, shall not affect any trace stream previously created. An initialized attributes object
29429 also serves to receive the attributes of an existing trace stream or trace log when calling the
29430 `posix_trace_get_attr()` function.

29431 RETURN VALUE

29432 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
29433 return the corresponding error number.

29434 ERRORS

29435 The `posix_trace_attr_destroy()` function may fail if:

29436 [EINVAL] The value of `attr` is invalid.

29437 The `posix_trace_attr_init()` function shall fail if:

29438 [ENOMEM] Insufficient memory exists to initialize the trace attributes object.

29439 EXAMPLES

29440 None.

29441 APPLICATION USAGE

29442 None.

29443 RATIONALE

29444 None.

29445 **FUTURE DIRECTIONS**

29446 None.

29447 **SEE ALSO**

29448 *posix_trace_create()*, *posix_trace_get_attr()*, *uname()*, the Base Definitions volume of
29449 IEEE Std 1003.1-2001, <**trace.h**>

29450 **CHANGE HISTORY**

29451 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29452 IEEE PASC Interpretation 1003.1 #123 is applied.

29453 NAME

29454 posix_trace_attr_getclockres, posix_trace_attr_getcreatetime, posix_trace_attr_getgenversion,
29455 posix_trace_attr_getname, posix_trace_attr_setname — retrieve and set information about a
29456 trace stream (**TRACING**)

29457 SYNOPSIS

```
29458 TRC    #include <time.h>
29459         #include <trace.h>

29460         int posix_trace_attr_getclockres(const trace_attr_t *attr,
29461         struct timespec *resolution);
29462         int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
29463         struct timespec *createtime);

29464         #include <trace.h>

29465         int posix_trace_attr_getgenversion(const trace_attr_t *attr,
29466         char *genversion);
29467         int posix_trace_attr_getname(const trace_attr_t *attr,
29468         char *tracename);
29469         int posix_trace_attr_setname(trace_attr_t *attr,
29470         const char *tracename);
29471
```

29472 DESCRIPTION

29473 The *posix_trace_attr_getclockres()* function shall copy the clock resolution of the clock used to
29474 generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the
29475 *attr* argument into the structure pointed to by the *resolution* argument.

29476 The *posix_trace_attr_getcreatetime()* function shall copy the trace stream creation time from the
29477 *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure
29478 pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of
29479 creation of the trace stream.

29480 The *posix_trace_attr_getgenversion()* function shall copy the string containing version information
29481 from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into
29482 the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of
29483 a character array which can store at least {TRACE_NAME_MAX} characters.

29484 The *posix_trace_attr_getname()* function shall copy the string containing the trace name from the
29485 *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string
29486 pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character
29487 array which can store at least {TRACE_NAME_MAX} characters.

29488 The *posix_trace_attr_setname()* function shall set the name in the *trace-name* attribute of the
29489 attributes object pointed to by the *attr* argument, using the trace name string supplied by the
29490 *tracename* argument. If the supplied string contains more than {TRACE_NAME_MAX}
29491 characters, the name copied into the *trace-name* attribute may be truncated to one less than the
29492 length of {TRACE_NAME_MAX} characters. The default value is a null string.

29493 RETURN VALUE

29494 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
29495 return the corresponding error number.

29496 If successful, the *posix_trace_attr_getclockres()* function stores the *clock-resolution* attribute value
29497 in the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

29498 If successful, the *posix_trace_attr_getcreatetime()* function stores the trace stream creation time in
29499 the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

29500 If successful, the *posix_trace_attr_getgenversion()* function stores the trace version information in
29501 the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

29502 If successful, the *posix_trace_attr_getname()* function stores the trace name in the string pointed
29503 to by *tracename*. Otherwise, the content of this string is unspecified.

29504 **ERRORS**

29505 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
29506 and *posix_trace_attr_getname()* functions may fail if:

29507 [EINVAL] The value specified by one of the arguments is invalid.

29508 **EXAMPLES**

29509 None.

29510 **APPLICATION USAGE**

29511 None.

29512 **RATIONALE**

29513 None.

29514 **FUTURE DIRECTIONS**

29515 None.

29516 **SEE ALSO**

29517 *posix_trace_attr_init()*, *posix_trace_create()*, *posix_trace_get_attr()*, *uname()*, the Base Definitions
29518 volume of IEEE Std 1003.1-2001, **<time.h>**, **<trace.h>**

29519 **CHANGE HISTORY**

29520 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29521 NAME

29522 posix_trace_attr_getinherited, posix_trace_attr_getlogfullpolicy,
29523 posix_trace_attr_getstreamfullpolicy, posix_trace_attr_setinherited,
29524 posix_trace_attr_setlogfullpolicy, posix_trace_attr_setstreamfullpolicy — retrieve and set the
29525 behavior of a trace stream (**TRACING**)

29526 SYNOPSIS

```
29527 TRC      #include <trace.h>

29528 TRC TRI   int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
29529                                     int *restrict inheritancepolicy);
29530 TRC TRL   int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
29531                                     int *restrict logpolicy);
29532 TRC      int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
29533                                     int *streampolicy);
29534 TRC TRI   int posix_trace_attr_setinherited(trace_attr_t *attr,
29535                                     int inheritancepolicy);
29536 TRC TRL   int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
29537                                     int logpolicy);
29538 TRC      int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
29539                                     int streampolicy);
29540
```

29541 DESCRIPTION

29542 TRI The *posix_trace_attr_getinherited()* and *posix_trace_attr_setinherited()* functions, respectively, shall
29543 get and set the inheritance policy stored in the *inheritance* attribute for traced processes across
29544 the *fork()* and *spawn()* operations. The *inheritance* attribute of the attributes object pointed to by
29545 the *attr* argument shall be set to one of the following values defined by manifest constants in the
29546 **<trace.h>** header:

29547 POSIX_TRACE_CLOSE_FOR_CHILD
29548 After a *fork()* or *spawn()* operation, the child shall not be traced, and tracing of the parent
29549 shall continue.

29550 POSIX_TRACE_INHERITED
29551 After a *fork()* or *spawn()* operation, if the parent is being traced, its child shall be
29552 concurrently traced using the same trace stream.

29553 The default value for the *inheritance* attribute is POSIX_TRACE_CLOSE_FOR_CHILD.

29554 TRL The *posix_trace_attr_getlogfullpolicy()* and *posix_trace_attr_setlogfullpolicy()* functions,
29555 respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the
29556 attributes object pointed to by the *attr* argument.

29557 The *log-full-policy* attribute shall be set to one of the following values defined by manifest
29558 constants in the **<trace.h>** header:

29559 POSIX_TRACE_LOOP
29560 The trace log shall loop until the associated trace stream is stopped. This policy means that
29561 when the trace log gets full, the file system shall reuse the resources allocated to the oldest
29562 trace events that were recorded. In this way, the trace log will always contain the most
29563 recent trace events flushed.

29564 POSIX_TRACE_UNTIL_FULL
29565 The trace stream shall be flushed to the trace log until the trace log is full. This condition can
29566 be deduced from the *posix_log_full_status* member status (see the **posix_trace_status_info**
29567 structure defined in **<trace.h>**). The last recorded trace event shall be the

29568	POSIX_TRACE_STOP trace event.
29569	POSIX_TRACE_APPEND
29570	The associated trace stream shall be flushed to the trace log without log size limitation. If
29571	the application specifies POSIX_TRACE_APPEND, the implementation shall ignore the
29572	<i>log-max-size</i> attribute.
29573	The default value for the <i>log-full-policy</i> attribute is POSIX_TRACE_LOOP.
29574	The <i>posix_trace_attr_getstreamfullpolicy()</i> and <i>posix_trace_attr_setstreamfullpolicy()</i> functions,
29575	respectively, shall get and set the trace stream full policy stored in the <i>stream-full-policy</i> attribute
29576	of the attributes object pointed to by the <i>attr</i> argument.
29577	The <i>stream-full-policy</i> attribute shall be set to one of the following values defined by manifest
29578	constants in the <trace.h> header:
29579	POSIX_TRACE_LOOP
29580	The trace stream shall loop until explicitly stopped by the <i>posix_trace_stop()</i> function. This
29581	policy means that when the trace stream is full, the trace system shall reuse the resources
29582	allocated to the oldest trace events recorded. In this way, the trace stream will always
29583	contain the most recent trace events recorded.
29584	POSIX_TRACE_UNTIL_FULL
29585	The trace stream will run until the trace stream resources are exhausted. Then the trace
29586	stream will stop. This condition can be deduced from <i>posix_stream_status</i> and
29587	<i>posix_stream_full_status</i> (see the posix_trace_status_info structure defined in <trace.h>).
29588	When this trace stream is read, a POSIX_TRACE_STOP trace event shall be reported after
29589	reporting the last recorded trace event. The trace system shall reuse the resources allocated
29590	to any trace events already reported—see the <i>posix_trace_getnext_event()</i> ,
29591	<i>posix_trace_trygetnext_event()</i> , and <i>posix_trace_timedgetnext_event()</i> functions—or already
29592	flushed for an active trace stream with log if the Trace Log option is supported; see the
29593	<i>posix_trace_flush()</i> function. The trace system shall restart the trace stream when it is empty
29594	and may restart it sooner. A POSIX_TRACE_START trace event shall be reported before
29595	reporting the next recorded trace event.
29596 TRL	POSIX_TRACE_FLUSH
29597	If the Trace Log option is supported, this policy is identical to the
29598	POSIX_TRACE_UNTIL_FULL trace stream full policy except that the trace stream shall be
29599	flushed regularly as if <i>posix_trace_flush()</i> had been explicitly called. Defining this policy for
29600	an active trace stream without log shall be invalid.
29601	The default value for the <i>stream-full-policy</i> attribute shall be POSIX_TRACE_LOOP for an active
29602	trace stream without log.
29603 TRL	If the Trace Log option is supported, the default value for the <i>stream-full-policy</i> attribute shall be
29604	POSIX_TRACE_FLUSH for an active trace stream with log.
29605	RETURN VALUE
29606	Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
29607	return the corresponding error number.
29608 TRI	If successful, the <i>posix_trace_attr_getinherited()</i> function shall store the <i>inheritance</i> attribute value
29609	in the object pointed to by <i>inheritancepolicy</i> . Otherwise, the content of this object is undefined.
29610 TRL	If successful, the <i>posix_trace_attr_getlogfullpolicy()</i> function shall store the <i>log-full-policy</i> attribute
29611	value in the object pointed to by <i>logpolicy</i> . Otherwise, the content of this object is undefined.
29612	If successful, the <i>posix_trace_attr_getstreamfullpolicy()</i> function shall store the <i>stream-full-policy</i>
29613	attribute value in the object pointed to by <i>streampolicy</i> . Otherwise, the content of this object is

29614 undefined.

29615 ERRORS

29616 These functions may fail if:

29617 [EINVAL] The value specified by at least one of the arguments is invalid.

29618 EXAMPLES

29619 None.

29620 APPLICATION USAGE

29621 None.

29622 RATIONALE

29623 None.

29624 FUTURE DIRECTIONS

29625 None.

29626 SEE ALSO

29627 *fork()*, *posix_trace_attr_init()*, *posix_trace_create()*, *posix_trace_flush()*, *posix_trace_get_attr()*,
 29628 *posix_trace_getnext_event()*, *posix_trace_start()*, *posix_trace_timedgetnext_event()*, the Base
 29629 Definitions volume of IEEE Std 1003.1-2001, <trace.h>

29630 CHANGE HISTORY

29631 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29632 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 39 is applied, adding the TRL and TRC |
 29633 margin codes to the *posix_trace_attr_setlogfullpolicy()* function. |

29634 **NAME**

29635 posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize,
 29636 posix_trace_attr_getmaxsystemeventsize, posix_trace_attr_getmaxusereventsize,
 29637 posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize,
 29638 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

29639 **SYNOPSIS**

```
29640 TRC    #include <sys/types.h>
29641         #include <trace.h>

29642 TRC TRL  int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
29643         size_t *restrict logsize);
29644 TRC      int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
29645         size_t *restrict maxdatasize);
29646      int posix_trace_attr_getmaxsystemeventsize(
29647         const trace_attr_t *restrict attr,
29648         size_t *restrict eventsize);
29649      int posix_trace_attr_getmaxusereventsize(
29650         const trace_attr_t *restrict attr,
29651         size_t data_len, size_t *restrict eventsize);
29652      int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
29653         size_t *restrict streamsize);
29654 TRC TRL  int posix_trace_attr_setlogsize(trace_attr_t *attr,
29655         size_t logsize);
29656 TRC      int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
29657         size_t maxdatasize);
29658      int posix_trace_attr_setstreamsize(trace_attr_t *attr,
29659         size_t streamsize);
29660
```

29661 **DESCRIPTION**

29662 TRL The *posix_trace_attr_getlogsize()* function shall copy the log size, in bytes, from the *log-max-size*
 29663 attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by
 29664 the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for
 29665 system and user trace events in the trace log. The default value for the *log-max-size* attribute is
 29666 implementation-defined.

29667 The *posix_trace_attr_setlogsize()* function shall set the maximum allowed size, in bytes, in the
 29668 *log-max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value
 29669 supplied by the *logsize* argument.

29670 The trace log size shall be used if the *log-full-policy* attribute is set to *POSIX_TRACE_LOOP* or
 29671 *POSIX_TRACE_UNTIL_FULL*. If the *log-full-policy* attribute is set to *POSIX_TRACE_APPEND*,
 29672 the implementation shall ignore the *log-max-size* attribute.

29673 The *posix_trace_attr_getmaxdatasize()* function shall copy the maximum user trace event data
 29674 size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr*
 29675 argument into the variable pointed to by the *maxdatasize* argument. The default value for the
 29676 *max-data-size* attribute is implementation-defined.

29677 The *posix_trace_attr_getmaxsystemeventsize()* function shall calculate the maximum memory size,
 29678 in bytes, required to store a single system trace event. This value is calculated for the trace
 29679 stream attributes object pointed to by the *attr* argument and is returned in the variable pointed
 29680 to by the *eventsize* argument.

29681 The values returned as the maximum memory sizes of the user and system trace events shall be
 29682 such that if the sum of the maximum memory sizes of a set of the trace events that may be
 29683 recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace
 29684 stream, the system provides the necessary resources for recording all those trace events, without
 29685 loss.

29686 The *posix_trace_attr_getmaxusereventsize()* function shall calculate the maximum memory size, in
 29687 bytes, required to store a single user trace event generated by a call to *posix_trace_event()* with a
 29688 *data_len* parameter equal to the *data_len* value specified in this call. This value is calculated for
 29689 the trace stream attributes object pointed to by the *attr* argument and is returned in the variable
 29690 pointed to by the *eventsize* argument.

29691 The *posix_trace_attr_getstreamsize()* function shall copy the stream size, in bytes, from the
 29692 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument into the variable
 29693 pointed to by the *streamsize* argument.

29694 This stream size is the current total memory size reserved for system and user trace events in the
 29695 trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The
 29696 stream size refers to memory used to store trace event records. Other stream data (for example,
 29697 trace attribute values) shall not be included in this size.

29698 The *posix_trace_attr_setmaxdatasize()* function shall set the maximum allowed size, in bytes, in
 29699 the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size
 29700 value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size
 29701 for the user data argument which may be passed to *posix_trace_event()*. The implementation
 29702 shall be allowed to truncate data passed to *trace_user_event* which is longer than *maxdatasize*.

29703 The *posix_trace_attr_setstreamsize()* function shall set the minimum allowed size, in bytes, in the
 29704 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size
 29705 value supplied by the *streamsize* argument.

29706 RETURN VALUE

29707 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 29708 return the corresponding error number.

29709 TRL The *posix_trace_attr_getlogsize()* function stores the maximum trace log allowed size in the object
 29710 pointed to by *logsize*, if successful.

29711 The *posix_trace_attr_getmaxdatasize()* function stores the maximum trace event record memory
 29712 size in the object pointed to by *maxdatasize*, if successful.

29713 The *posix_trace_attr_getmaxsystemeventsize()* function stores the maximum memory size to store
 29714 a single system trace event in the object pointed to by *eventsize*, if successful.

29715 The *posix_trace_attr_getmaxusereventsize()* function stores the maximum memory size to store a
 29716 single user trace event in the object pointed to by *eventsize*, if successful.

29717 The *posix_trace_attr_getstreamsize()* function stores the maximum trace stream allowed size in
 29718 the object pointed to by *streamsize*, if successful.

29719 ERRORS

29720 These functions may fail if:

29721 [EINVAL] The value specified by one of the arguments is invalid.

29722 **EXAMPLES**

29723 None.

29724 **APPLICATION USAGE**

29725 None.

29726 **RATIONALE**

29727 None.

29728 **FUTURE DIRECTIONS**

29729 None.

29730 **SEE ALSO**29731 *posix_trace_attr_init()*, *posix_trace_create()*, *posix_trace_event()*, *posix_trace_get_attr()*, the Base29732 Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<trace.h>**29733 **CHANGE HISTORY**

29734 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29735 **NAME**

29736 posix_trace_attr_getname — retrieve and set information about a trace stream (**TRACING**)

29737 **SYNOPSIS**

29738 TRC `#include <trace.h>`

```
29739        int posix_trace_attr_getname(const trace_attr_t *attr,  
29740                                    char *tracename);
```

29741

29742 **DESCRIPTION**

29743 Refer to *posix_trace_attr_getclockres()*.

29744 **NAME**

29745 posix_trace_attr_getstreamfullpolicy — retrieve and set the behavior of a trace stream
29746 **(TRACING)**

29747 **SYNOPSIS**

29748 TRC #include <trace.h>

```
29749       int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,  
29750                                           int *streampolicy);
```

29751

29752 **DESCRIPTION**

29753 Refer to *posix_trace_attr_getinherited()*.

29754 NAME

29755 posix_trace_attr_getstreamsize — retrieve and set trace stream size attributes (**TRACING**)

29756 SYNOPSIS

29757 TRC #include <sys/types.h>

29758 #include <trace.h>

```
29759        int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,  
29760                                          size_t *restrict streamsize);
```

29761

29762 DESCRIPTION

29763 Refer to *posix_trace_attr_getlogsize()*.

29764 **NAME**

29765 posix_trace_attr_init — initialize the trace stream attributes object (**TRACING**)

29766 **SYNOPSIS**

29767 TRC #include <trace.h>

29768 int posix_trace_attr_init(trace_attr_t *attr);

29769

29770 **DESCRIPTION**

29771 Refer to *posix_trace_attr_destroy()*.

29772 NAME

29773 posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy — retrieve and set the behavior
29774 of a trace stream (**TRACING**)

29775 SYNOPSIS

```
29776 TRC      #include <trace.h>
29777 TRC TRI  int posix_trace_attr_setinherited(trace_attr_t *attr,
29778                                     int inheritancepolicy);
29779 TRC TRL  int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
29780                                     int logpolicy);
29781
```

29782 DESCRIPTION

29783 Refer to *posix_trace_attr_getinherited()*.

29784 **NAME**

29785 posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize — retrieve and set trace stream
29786 size attributes (**TRACING**)

29787 **SYNOPSIS**

29788 TRC #include <sys/types.h>

29789 #include <trace.h>

29790 TRC TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
29791 size_t logsize);

29792 TRC int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
29793 size_t maxdatasize);

29794

29795 **DESCRIPTION**

29796 Refer to *posix_trace_attr_getlogsize()*.

29797 **NAME**

29798 posix_trace_attr_setname — retrieve and set information about a trace stream (**TRACING**)

29799 **SYNOPSIS**

29800 TRC `#include <trace.h>`

29801 `int posix_trace_attr_setname(trace_attr_t *attr,`
29802 `const char *tracename);`

29803

29804 **DESCRIPTION**

29805 Refer to *posix_trace_attr_getclockres()*.

29806 **NAME**

29807 posix_trace_attr_setstreamfullpolicy — retrieve and set the behavior of a trace stream
29808 **(TRACING)**

29809 **SYNOPSIS**

29810 TRC #include <trace.h>

29811 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
29812 int streampolicy);

29813

29814 **DESCRIPTION**

29815 Refer to *posix_trace_attr_getinherited()*.

29816 NAME

29817 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

29818 SYNOPSIS

29819 TRC #include <sys/types.h>

29820 #include <trace.h>

29821 int posix_trace_attr_setstreamsize(trace_attr_t *attr,
29822 size_t streamsize);

29823

29824 DESCRIPTION

29825 Refer to *posix_trace_attr_getlogsize()*.

29826 **NAME**29827 posix_trace_clear — clear trace stream and trace log (**TRACING**)29828 **SYNOPSIS**

29829 TRC #include <sys/types.h>

29830 #include <trace.h>

29831 int posix_trace_clear(trace_id_t trid);

29832

29833 **DESCRIPTION**

29834 The *posix_trace_clear()* function shall reinitialize the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create()* function, except that the same allocated resources shall be reused, the mapping of trace event type identifiers to trace event names shall be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

29839 All trace events in the trace stream recorded before the call to *posix_trace_clear()* shall be lost. The *posix_stream_full_status* status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that all trace events that occurred during the *posix_trace_clear()* call are recorded; the behavior with respect to trace points that may occur during this call is unspecified.

29843 TRL If the Trace Log option is supported and the trace stream has been created with a log, the *posix_trace_clear()* function shall reinitialize the trace stream with the same behavior as if the trace stream was created without the log, plus it shall reinitialize the trace log associated with the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create_withlog()* function, except that the same allocated resources, for the trace log, may be reused and the associated trace stream status remains unchanged. The first trace event recorded in the trace log after the call to *posix_trace_clear()* shall be the same as the first trace event recorded in the active trace stream after the call to *posix_trace_clear()*. The *posix_log_full_status* status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that all trace events that occurred during the *posix_trace_clear()* call are recorded in the trace log; the behavior with respect to trace points that may occur during this call is unspecified. If the log full policy is `POSIX_TRACE_APPEND`, the effect of a call to this function is unspecified for the trace log associated with the trace stream identified by the *trid* argument.

29856 **RETURN VALUE**

29857 Upon successful completion, the *posix_trace_clear()* function shall return a value of zero. Otherwise, it shall return the corresponding error number.

29859 **ERRORS**29860 The *posix_trace_clear()* function shall fail if:29861 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream.29862 **EXAMPLES**

29863 None.

29864 **APPLICATION USAGE**

29865 None.

29866 **RATIONALE**

29867 None.

29868 **FUTURE DIRECTIONS**

29869 None.

29870 SEE ALSO

29871 *posix_trace_attr_init()*, *posix_trace_create()*, *posix_trace_flush()*, *posix_trace_get_attr()*, the Base
29872 Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<trace.h>**

29873 CHANGE HISTORY

29874 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29875 IEEE PASC Interpretation 1003.1 #123 is applied.

29876 **NAME**29877 posix_trace_close, posix_trace_open, posix_trace_rewind — trace log management (**TRACING**)29878 **SYNOPSIS**

29879 TRC TRL #include <trace.h>

```

29880        int posix_trace_close(trace_id_t trid);
29881        int posix_trace_open(int file_desc, trace_id_t *trid);
29882        int posix_trace_rewind(trace_id_t trid);
29883 
```

29884 **DESCRIPTION**

29885 The *posix_trace_close()* function shall deallocate the trace log identifier indicated by *trid*, and all
 29886 of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall
 29887 fail.

29888 The *posix_trace_open()* function shall allocate the necessary resources and establish the
 29889 connection between a trace log identified by the *file_desc* argument and a trace stream identifier
 29890 identified by the object pointed to by the *trid* argument. The *file_desc* argument should be a valid
 29891 open file descriptor that corresponds to a trace log. The *file_desc* argument shall be open for
 29892 reading. The current trace event timestamp, which specifies the timestamp of the trace event
 29893 that will be read by the next call to *posix_trace_getnext_event()*, shall be set to the timestamp of
 29894 the oldest trace event recorded in the trace log identified by *trid*.

29895 The *posix_trace_open()* function shall return a trace stream identifier in the variable pointed to by
 29896 the *trid* argument, that may only be used by the following functions:

29897 <i>posix_trace_close()</i>	<i>posix_trace_get_attr()</i>
29898 <i>posix_trace_eventid_equal()</i>	<i>posix_trace_get_status()</i>
29899 <i>posix_trace_eventid_get_name()</i>	<i>posix_trace_getnext_event()</i>
29900 <i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_rewind()</i>
29901 <i>posix_trace_eventtypelist_rewind()</i>	

29902 In particular, notice that the operations normally used by a trace controller process, such as
 29903 *posix_trace_start()*, *posix_trace_stop()*, or *posix_trace_shutdown()*, cannot be invoked using the
 29904 trace stream identifier returned by the *posix_trace_open()* function.

29905 The *posix_trace_rewind()* function shall reset the current trace event timestamp, which specifies
 29906 the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event()*,
 29907 to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

29908 **RETURN VALUE**

29909 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 29910 return the corresponding error number.

29911 If successful, the *posix_trace_open()* function stores the trace stream identifier value in the object
 29912 pointed to by *trid*.

29913 **ERRORS**

29914 The *posix_trace_open()* function shall fail if:

29915 [EINTR]	The operation was interrupted by a signal and thus no trace log was opened.
29916 [EINVAL]	The object pointed to by <i>file_desc</i> does not correspond to a valid trace log.

29917 The *posix_trace_close()* and *posix_trace_rewind()* functions may fail if:

29918 [EINVAL]	The object pointed to by <i>trid</i> does not correspond to a valid trace log.

29919 **EXAMPLES**

29920 None.

29921 **APPLICATION USAGE**

29922 None.

29923 **RATIONALE**

29924 None.

29925 **FUTURE DIRECTIONS**

29926 None.

29927 **SEE ALSO**

29928 *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_getnext_event()*, the Base Definitions
29929 volume of IEEE Std 1003.1-2001, <**trace.h**>

29930 **CHANGE HISTORY**

29931 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29932 IEEE PASC Interpretation 1003.1 #123 is applied.

29933 **NAME**

29934 posix_trace_create, posix_trace_create_withlog, posix_trace_flush, posix_trace_shutdown —
 29935 trace stream initialization, flush, and shutdown from a process (**TRACING**)

29936 **SYNOPSIS**

```
29937 TRC    #include <sys/types.h>
29938        #include <trace.h>

29939        int posix_trace_create(pid_t pid,
29940                               const trace_attr_t *restrict attr,
29941                               trace_id_t *restrict trid);
29942 TRC TRL int posix_trace_create_withlog(pid_t pid,
29943                                         const trace_attr_t *restrict attr, int file_desc,
29944                                         trace_id_t *restrict trid);
29945        int posix_trace_flush(trace_id_t trid);
29946 TRC        int posix_trace_shutdown(trace_id_t trid);
29947
```

29948 **DESCRIPTION**

29949 The *posix_trace_create()* function shall create an active trace stream. It allocates all the resources
 29950 needed by the trace stream being created for tracing the process specified by *pid* in accordance
 29951 with the *attr* argument. The *attr* argument represents the initial attributes of the trace stream and
 29952 shall have been initialized by the function *posix_trace_attr_init()* prior to the *posix_trace_create()*
 29953 call. If the argument *attr* is NULL, the default attributes shall be used. The *attr* attributes object
 29954 shall be manipulated through a set of functions described in the *posix_trace_attr* family of
 29955 functions. If the attributes of the object pointed to by *attr* are modified later, the attributes of the
 29956 trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream
 29957 shall be set to the value of the system clock, if the Timers option is not supported, or to the value
 29958 of the CLOCK_REALTIME clock, if the Timers option is supported.

29959 The *pid* argument represents the target process to be traced. If the process executing this
 29960 function does not have appropriate privileges to trace the process identified by *pid*, an error shall
 29961 be returned. If the *pid* argument is zero, the calling process shall be traced.

29962 The *posix_trace_create()* function shall store the trace stream identifier of the new trace stream in
 29963 the object pointed to by the *trid* argument. This trace stream identifier shall be used in
 29964 subsequent calls to control tracing. The *trid* argument may only be used by the following
 29965 functions:

29966	<i>posix_trace_clear()</i>	<i>posix_trace_getnext_event()</i>
29967	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_shutdown()</i>
29968	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_start()</i>
29969	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_stop()</i>
29970	<i>posix_trace_eventtypelist_rewind()</i>	<i>posix_trace_timedgetnext_event()</i>
29971	<i>posix_trace_get_attr()</i>	<i>posix_trace_trid_eventid_open()</i>
29972	<i>posix_trace_get_status()</i>	<i>posix_trace_trygetnext_event()</i>

29973 TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*
 29974 argument:

29975 *posix_trace_get_filter()* *posix_trace_set_filter()*

29976

29977 In particular, notice that the operations normally used by a trace analyzer process, such as
 29978 *posix_trace_rewind()* or *posix_trace_close()*, cannot be invoked using the trace stream identifier
 29979 returned by the *posix_trace_create()* function.

29980 TEF A trace stream shall be created in a suspended state. If the Trace Event Filter option is
 29981 supported, its trace event type filter shall be empty.

29982 The *posix_trace_create()* function may be called multiple times from the same or different
 29983 processes, with the system-wide limit indicated by the runtime invariant value
 29984 {TRACE_SYS_MAX}, which has the minimum value {_POSIX_TRACE_SYS_MAX}.

29985 The trace stream identifier returned by the *posix_trace_create()* function in the argument pointed
 29986 to by *trid* is valid only in the process that made the function call. If it is used from another
 29987 process, that is a child process, in functions defined in IEEE Std 1003.1-2001, these functions shall
 29988 return with the error [EINVAL].

29989 TRL The *posix_trace_create_withlog()* function shall be equivalent to *posix_trace_create()*, except that it
 29990 associates a trace log with this stream. The *file_desc* argument shall be the file descriptor
 29991 designating the trace log destination. The function shall fail if this file descriptor refers to a file
 29992 with a file type that is not compatible with the log policy associated with the trace log. The list of
 29993 the appropriate file types that are compatible with each log policy is implementation-defined.

29994 The *posix_trace_create_withlog()* function shall return in the parameter pointed to by *trid* the trace
 29995 stream identifier, which uniquely identifies the newly created trace stream, and shall be used in
 29996 subsequent calls to control tracing. The *trid* argument may only be used by the following
 29997 functions:

29998	<i>posix_trace_clear()</i>	<i>posix_trace_getnext_event()</i>
29999	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_shutdown()</i>
30000	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_start()</i>
30001	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_stop()</i>
30002	<i>posix_trace_eventtypelist_rewind()</i>	<i>posix_trace_timedgetnext_event()</i>
30003	<i>posix_trace_flush()</i>	<i>posix_trace_trid_eventid_open()</i>
30004	<i>posix_trace_get_attr()</i>	<i>posix_trace_trygetnext_event()</i>
30005	<i>posix_trace_get_status()</i>	

30006

30007 TRL TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*
 30008 argument:

30009 *posix_trace_get_filter()* *posix_trace_set_filter()*

30010

30011 TRL In particular, notice that the operations normally used by a trace analyzer process, such as
 30012 *posix_trace_rewind()* or *posix_trace_close()*, cannot be invoked using the trace stream identifier
 30013 returned by the *posix_trace_create_withlog()* function.

30014 The *posix_trace_flush()* function shall initiate a flush operation which copies the contents of the
 30015 trace stream identified by the argument *trid* into the trace log associated with the trace stream at
 30016 the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this
 30017 function shall return an error. The termination of the flush operation can be polled by the
 30018 *posix_trace_get_status()* function. During the flush operation, it shall be possible to trace new
 30019 trace events up to the point when the trace stream becomes full. After flushing is completed, the
 30020 space used by the flushed trace events shall be available for tracing new trace events.

30021	If flushing the trace stream causes the resulting trace log to become full, the trace log full policy shall be applied. If the trace <i>log-full-policy</i> attribute is set, the following occurs:	
30022		
30023	POSIX_TRACE_UNTIL_FULL	
30024	The trace events that have not yet been flushed shall be discarded.	
30025	POSIX_TRACE_LOOP	
30026	The trace events that have not yet been flushed shall be written to the beginning of the trace log, overwriting previous trace events stored there.	
30027		
30028	POSIX_TRACE_APPEND	
30029	The trace events that have not yet been flushed shall be appended to the trace log.	
30030		
30031	The <i>posix_trace_shutdown()</i> function shall stop the tracing of trace events in the trace stream identified by <i>trid</i> , as if <i>posix_trace_stop()</i> had been invoked. The <i>posix_trace_shutdown()</i> function shall free all the resources associated with the trace stream.	
30032		
30033		
30034	The <i>posix_trace_shutdown()</i> function shall not return until all the resources associated with the trace stream have been freed. When the <i>posix_trace_shutdown()</i> function returns, the <i>trid</i> argument becomes an invalid trace stream identifier. A call to this function shall unconditionally deallocate the resources regardless of whether all trace events have been retrieved by the analyzer process. Any thread blocked on one of the <i>trace_getnext_event()</i> functions (which specified this <i>trid</i>) before this call is unblocked with the error [EINVAL].	
30035		
30036		
30037		
30038		
30039		
30040	If the process exits, invokes a member of the <i>exec</i> family of functions, or is terminated, the trace streams that the process had created and that have not yet been shut down, shall be automatically shut down as if an explicit call were made to the <i>posix_trace_shutdown()</i> function.	
30041		
30042		
30043	TRL	For an active trace stream with log, when the <i>posix_trace_shutdown()</i> function is called, all trace events that have not yet been flushed to the trace log shall be flushed, as in the <i>posix_trace_flush()</i> function, and the trace log shall be closed.
30044		
30045		
30046	When a trace log is closed, all the information that may be retrieved later from the trace log through the trace interface shall have been written to the trace log. This information includes the trace attributes, the list of trace event types (with the mapping between trace event names and trace event type identifiers), and the trace status.	
30047		
30048		
30049		
30050	In addition, unspecified information shall be written to the trace log to allow detection of a valid trace log during the <i>posix_trace_open()</i> operation.	
30051		
30052	The <i>posix_trace_shutdown()</i> function shall not return until all trace events have been flushed.	
30053	RETURN VALUE	
30054	Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.	
30055		
30056	TRL	The <i>posix_trace_create()</i> and <i>posix_trace_create_withlog()</i> functions store the trace stream identifier value in the object pointed to by <i>trid</i> , if successful.
30057		
30058	ERRORS	
30059	TRL	The <i>posix_trace_create()</i> and <i>posix_trace_create_withlog()</i> functions shall fail if:
30060	[EAGAIN]	No more trace streams can be started now. {TRACE_SYS_MAX} has been exceeded.
30061		
30062	[EINTR]	The operation was interrupted by a signal. No trace stream was created.
30063	[EINVAL]	One or more of the trace parameters specified by the <i>attr</i> parameter is invalid.

30064	[ENOMEM]	The implementation does not currently have sufficient memory to create the trace stream with the specified parameters.
30065		
30066	[EPERM]	The caller does not have appropriate privilege to trace the process specified by <i>pid</i> .
30067		
30068	[ESRCH]	The <i>pid</i> argument does not refer to an existing process.
30069	TRL	The <i>posix_trace_create_withlog()</i> function shall fail if:
30070	[EBADF]	The <i>file_desc</i> argument is not a valid file descriptor open for writing.
30071	[EINVAL]	The <i>file_desc</i> argument refers to a file with a file type that does not support the log policy associated with the trace log.
30072		
30073	[ENOSPC]	No space left on device. The device corresponding to the argument <i>file_desc</i> does not contain the space required to create this trace log.
30074		
30075		
30076	TRL	The <i>posix_trace_flush()</i> and <i>posix_trace_shutdown()</i> functions shall fail if:
30077	[EINVAL]	The value of the <i>trid</i> argument does not correspond to an active trace stream with log.
30078		
30079	[EFBIG]	The trace log file has attempted to exceed an implementation-defined maximum file size.
30080		
30081	[ENOSPC]	No space left on device.
30082		

30083 EXAMPLES

30084 None.

30085 APPLICATION USAGE

30086 None.

30087 RATIONALE

30088 None.

30089 FUTURE DIRECTIONS

30090 None.

30091 SEE ALSO

30092 *clock_getres()*, *exec*, *posix_trace_attr_init()*, *posix_trace_clear()*, *posix_trace_close()*,
 30093 *posix_trace_eventid_equal()*, *posix_trace_eventtypelist_getnext_id()*, *posix_trace_flush()*,
 30094 *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_get_status()*, *posix_trace_getnext_event()*,
 30095 *posix_trace_open()*, *posix_trace_set_filter()*, *posix_trace_shutdown()*, *posix_trace_start()*,
 30096 *posix_trace_timedgetnext_event()*, *posix_trace_trid_eventid_open()*, *posix_trace_start()*, *time()*, the
 30097 Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <trace.h>

30098 CHANGE HISTORY

30099 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30100 **NAME**

30101 posix_trace_event, posix_trace_eventid_open — trace functions for instrumenting application
 30102 code (**TRACING**)

30103 **SYNOPSIS**

30104 TRC #include <sys/types.h>

30105 #include <trace.h>

```
30106 void posix_trace_event(trace_event_id_t event_id,
30107     const void *restrict data_ptr, size_t data_len);
30108 int posix_trace_eventid_open(const char *restrict event_name,
30109     trace_event_id_t *restrict event_id);
30110
```

30111 **DESCRIPTION**

30112 The *posix_trace_event()* function shall record the *event_id* and the user data pointed to by *data_ptr*
 30113 in the trace stream into which the calling process is being traced and in which *event_id* is not
 30114 filtered out. If the total size of the user trace event data represented by *data_len* is not greater
 30115 than the declared maximum size for user trace event data, then the *truncation-status* attribute of
 30116 the trace event recorded is POSIX_TRACE_NOT_TRUNCATED. Otherwise, the user trace event
 30117 data is truncated to this declared maximum size and the *truncation-status* attribute of the trace
 30118 event recorded is POSIX_TRACE_TRUNCATED_RECORD.

30119 If there is no trace stream created for the process or if the created trace stream is not running, or
 30120 if the trace event specified by *event_id* is filtered out in the trace stream, the *posix_trace_event()*
 30121 function shall have no effect.

30122 The *posix_trace_eventid_open()* function shall associate a user trace event name with a trace event
 30123 type identifier for the calling process. The trace event name is the string pointed to by the
 30124 argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters
 30125 (which has the minimum value {POSIX_TRACE_EVENT_NAME_MAX}). The number of user
 30126 trace event type identifiers that can be defined for any given process is limited by the maximum
 30127 value {TRACE_USER_EVENT_MAX}, which has the minimum value
 30128 {POSIX_TRACE_USER_EVENT_MAX}.

30129 If the Trace Inherit option is not supported, the *posix_trace_eventid_open()* function shall
 30130 associate the user trace event name pointed to by the *event_name* argument with a trace event
 30131 type identifier that is unique for the traced process, and is returned in the variable pointed to by
 30132 the *event_id* argument. If the user trace event name has already been mapped for the traced
 30133 process, then the previously assigned trace event type identifier shall be returned. If the per-
 30134 process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been
 30135 reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7 (on page 79))
 30136 user trace event shall be returned.

30137 TRI If the Trace Inherit option is supported, the *posix_trace_eventid_open()* function shall associate the
 30138 user trace event name pointed to by the *event_name* argument with a trace event type identifier
 30139 that is unique for all the processes being traced in this same trace stream, and is returned in the
 30140 variable pointed to by the *event_id* argument. If the user trace event name has already been
 30141 mapped for the traced processes, then the previously assigned trace event type identifier shall be
 30142 returned. If the per-process user trace event name limit represented by
 30143 {TRACE_USER_EVENT_MAX} has been reached, the pre-defined
 30144 POSIX_TRACE_UNNAMED_USEREVENT (Table 2-7 (on page 79)) user trace event shall be
 30145 returned.

30146 **Note:** The above procedure, together with the fact that multiple processes can only be traced into the
 30147 same trace stream by inheritance, ensure that all the processes that are traced into a trace
 30148 stream have the same mapping of trace event names to trace event type identifiers.

30149

30150 If there is no trace stream created, the *posix_trace_eventid_open()* function shall store this
30151 information for future trace streams created for this process.

30152 RETURN VALUE

30153 No return value is defined for the *posix_trace_event()* function.

30154 Upon successful completion, the *posix_trace_eventid_open()* function shall return a value of zero.
30155 Otherwise, it shall return the corresponding error number. The *posix_trace_eventid_open()*
30156 function stores the trace event type identifier value in the object pointed to by *event_id*, if
30157 successful.

30158 ERRORS

30159 The *posix_trace_eventid_open()* function shall fail if:

30160 [ENAMETOOLONG]

30161 The size of the name pointed to by the *event_name* argument was longer than
30162 the implementation-defined value {TRACE_EVENT_NAME_MAX}.

30163 EXAMPLES

30164 None.

30165 APPLICATION USAGE

30166 None.

30167 RATIONALE

30168 None.

30169 FUTURE DIRECTIONS

30170 None.

30171 SEE ALSO

30172 Table 2-7 (on page 79), *posix_trace_start()*, *posix_trace_trid_eventid_open()*, the Base Definitions
30173 volume of IEEE Std 1003.1-2001, <sys/types.h>, <trace.h>

30174 CHANGE HISTORY

30175 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30176 IEEE PASC Interpretation 1003.1 #123 is applied.

30177 IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of
30178 the *posix_trace_eventid_open()* function and the *event_id* argument.

30179 **NAME**

30180 posix_trace_eventid_equal, posix_trace_eventid_get_name, posix_trace_trid_eventid_open —
 30181 manipulate the trace event type identifier (**TRACING**)

30182 **SYNOPSIS**

```
30183 TRC      #include <trace.h>

30184      int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
30185                                  trace_event_id_t event2);
30186      int posix_trace_eventid_get_name(trace_id_t trid,
30187                                     trace_event_id_t event, char *event_name);
30188 TRC TEF   int posix_trace_trid_eventid_open(trace_id_t trid,
30189                                             const char *restrict event_name,
30190                                             trace_event_id_t *restrict event);
30191
```

30192 **DESCRIPTION**

30193 The *posix_trace_eventid_equal()* function shall compare the trace event type identifiers *event1* and
 30194 *event2* from the same trace stream or the same trace log identified by the *trid* argument. If the
 30195 trace event type identifiers *event1* and *event2* are from different trace streams, the return value
 30196 shall be unspecified.

30197 The *posix_trace_eventid_get_name()* function shall return, in the argument pointed to by
 30198 *event_name*, the trace event name associated with the trace event type identifier identified by the
 30199 argument *event*, for the trace stream or for the trace log identified by the *trid* argument. The
 30200 name of the trace event shall have a maximum of {TRACE_EVENT_NAME_MAX} characters
 30201 (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). Successive calls to
 30202 this function with the same trace event type identifier and the same trace stream identifier shall
 30203 return the same event name.

30204 TEF The *posix_trace_trid_eventid_open()* function shall associate a user trace event name with a trace
 30205 event type identifier for a given trace stream. The trace stream is identified by the *trid* argument,
 30206 and it shall be an active trace stream. The trace event name is the string pointed to by the
 30207 argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters
 30208 (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). The number of user
 30209 trace event type identifiers that can be defined for any given process is limited by the maximum
 30210 value {TRACE_USER_EVENT_MAX}, which has the minimum value
 30211 {_POSIX_TRACE_USER_EVENT_MAX}.

30212 If the Trace Inherit option is not supported, the *posix_trace_trid_eventid_open()* function shall
 30213 associate the user trace event name pointed to by the *event_name* argument with a trace event
 30214 type identifier that is unique for the process being traced in the trace stream identified by the *trid*
 30215 argument, and is returned in the variable pointed to by the *event* argument. If the user trace
 30216 event name has already been mapped for the traced process, then the previously assigned trace
 30217 event type identifier shall be returned. If the per-process user trace event name limit represented
 30218 by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined
 30219 POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7 (on page 79)) user trace event shall be
 30220 returned.

30221 TEF TRI If the Trace Inherit option is supported, the *posix_trace_trid_eventid_open()* function shall
 30222 associate the user trace event name pointed to by the *event_name* argument with a trace event
 30223 type identifier that is unique for all the processes being traced in the trace stream identified by
 30224 the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user
 30225 trace event name has already been mapped for the traced processes, then the previously
 30226 assigned trace event type identifier shall be returned. If the per-process user trace event name
 30227 limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined

30228 POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7 (on page 79)) user trace event shall be
30229 returned.

30230 RETURN VALUE

30231 TEF Upon successful completion, the *posix_trace_eventid_get_name()* and
30232 *posix_trace_trid_eventid_open()* functions shall return a value of zero. Otherwise, they shall return
30233 the corresponding error number.

30234 The *posix_trace_eventid_equal()* function shall return a non-zero value if *event1* and *event2* are
30235 equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or
30236 *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid*
30237 is invalid, the behavior shall be unspecified.

30238 The *posix_trace_eventid_get_name()* function stores the trace event name value in the object
30239 pointed to by *event_name*, if successful.

30240 TEF The *posix_trace_trid_eventid_open()* function stores the trace event type identifier value in the
30241 object pointed to by *event*, if successful.

30242 ERRORS

30243 TEF The *posix_trace_eventid_get_name()* and *posix_trace_trid_eventid_open()* functions shall fail if:

30244 [EINVAL] The *trid* argument was not a valid trace stream identifier.

30245 TEF The *posix_trace_trid_eventid_open()* function shall fail if:

30246 TEF [ENAMETOOLONG]

30247 The size of the name pointed to by the *event_name* argument was longer than
30248 the implementation-defined value {TRACE_EVENT_NAME_MAX}.

30249 The *posix_trace_eventid_get_name()* function shall fail if:

30250 [EINVAL] The trace event type identifier *event* was not associated with any name.

30251 EXAMPLES

30252 None.

30253 APPLICATION USAGE

30254 None.

30255 RATIONALE

30256 None.

30257 FUTURE DIRECTIONS

30258 None.

30259 SEE ALSO

30260 Table 2-7 (on page 79), *posix_trace_event()*, *posix_trace_getnext_event()*, the Base Definitions
30261 volume of IEEE Std 1003.1-2001, <trace.h>

30262 CHANGE HISTORY

30263 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30264 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

30265 **NAME**

30266 posix_trace_eventid_open — trace functions for instrumenting application code (**TRACING**)

30267 **SYNOPSIS**

30268 TRC #include <sys/types.h>

30269 #include <trace.h>

30270 int posix_trace_eventid_open(const char *restrict event_name,
30271 trace_event_id_t *restrict event_id);

30272

30273 **DESCRIPTION**

30274 Refer to *posix_trace_event()*.

30275 NAME

30276 posix_trace_eventset_add, posix_trace_eventset_del, posix_trace_eventset_empty,
30277 posix_trace_eventset_fill, posix_trace_eventset_ismember — manipulate trace event type sets
30278 (TRACING)

30279 SYNOPSIS

```
30280 TRC TEF #include <trace.h>

30281 int posix_trace_eventset_add(trace_event_id_t event_id,
30282     trace_event_set_t *set);
30283 int posix_trace_eventset_del(trace_event_id_t event_id,
30284     trace_event_set_t *set);
30285 int posix_trace_eventset_empty(trace_event_set_t *set);
30286 int posix_trace_eventset_fill(trace_event_set_t *set, int what);
30287 int posix_trace_eventset_ismember(trace_event_id_t event_id,
30288     const trace_event_set_t *restrict set, int *restrict ismember);
30289
```

30290 DESCRIPTION

30291 These primitives manipulate sets of trace event types. They operate on data objects addressable
30292 by the application, not on the current trace event filter of any trace stream.

30293 The *posix_trace_eventset_add()* and *posix_trace_eventset_del()* functions, respectively, shall add or
30294 delete the individual trace event type specified by the value of the argument *event_id* to or from
30295 the trace event type set pointed to by the argument *set*. Adding a trace event type already in the
30296 set or deleting a trace event type not in the set shall not be considered an error.

30297 The *posix_trace_eventset_empty()* function shall initialize the trace event type set pointed to by
30298 the *set* argument such that all trace event types defined, both system and user, shall be excluded
30299 from the set.

30300 The *posix_trace_eventset_fill()* function shall initialize the trace event type set pointed to by the
30301 argument *set*, such that the set of trace event types defined by the argument *what* shall be
30302 included in the set. The value of the argument *what* shall consist of one of the following values,
30303 as defined in the **<trace.h>** header:

30304 POSIX_TRACE_WOPID_EVENTS

30305 All the process-independent implementation-defined system trace event types are included
30306 in the set.

30307 POSIX_TRACE_SYSTEM_EVENTS

30308 All the implementation-defined system trace event types are included in the set, as are those
30309 defined in IEEE Std 1003.1-2001.

30310 POSIX_TRACE_ALL_EVENTS

30311 All trace event types defined, both system and user, are included in the set.

30312 Applications shall call either *posix_trace_eventset_empty()* or *posix_trace_eventset_fill()* at least
30313 once for each object of type **trace_event_set_t** prior to any other use of that object. If such an
30314 object is not initialized in this way, but is nonetheless supplied as an argument to any of the
30315 *posix_trace_eventset_add()*, *posix_trace_eventset_del()*, or *posix_trace_eventset_ismember()* functions,
30316 the results are undefined.

30317 The *posix_trace_eventset_ismember()* function shall test whether the trace event type specified by
30318 the value of the argument *event_id* is a member of the set pointed to by the argument *set*. The
30319 value returned in the object pointed to by *ismember* argument is zero if the trace event type
30320 identifier is not a member of the set and a value different from zero if it is a member of the set.

30321 RETURN VALUE

30322 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30323 return the corresponding error number.

30324 ERRORS

30325 These functions may fail if:

30326 [EINVAL] The value of one of the arguments is invalid.

30327 EXAMPLES

30328 None.

30329 APPLICATION USAGE

30330 None.

30331 RATIONALE

30332 None.

30333 FUTURE DIRECTIONS

30334 None.

30335 SEE ALSO

30336 *posix_trace_set_filter()*, *posix_trace_trid_eventid_open()*, the Base Definitions volume of
30337 IEEE Std 1003.1-2001, <**trace.h**>

30338 CHANGE HISTORY

30339 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30340 NAME

30341 posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind — iterate over a
30342 mapping of trace event types (**TRACING**)

30343 SYNOPSIS

```
30344 TRC    #include <trace.h>

30345    int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
30346      trace_event_id_t *restrict event, int *restrict unavailable);
30347    int posix_trace_eventtypelist_rewind(trace_id_t trid);
30348
```

30349 DESCRIPTION

30350 The first time *posix_trace_eventtypelist_getnext_id()* is called, the function shall return in the
30351 variable pointed to by *event* the first trace event type identifier of the list of trace events of the
30352 trace stream identified by the *trid* argument. Successive calls to
30353 *posix_trace_eventtypelist_getnext_id()* return in the variable pointed to by *event* the next trace
30354 event type identifier in that same list. Each time a trace event type identifier is successfully
30355 written into the variable pointed to by the *event* argument, the variable pointed to by the
30356 *unavailable* argument shall be set to zero. When no more trace event type identifiers are
30357 available, and so none is returned, the variable pointed to by the *unavailable* argument shall be
30358 set to a value different from zero.

30359 The *posix_trace_eventtypelist_rewind()* function shall reset the next trace event type identifier to
30360 be read to the first trace event type identifier from the list of trace events used in the trace stream
30361 identified by *trid*.

30362 RETURN VALUE

30363 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30364 return the corresponding error number.

30365 The *posix_trace_eventtypelist_getnext_id()* function stores the trace event type identifier value in
30366 the object pointed to by *event*, if successful.

30367 ERRORS

30368 These functions shall fail if:

30369 [EINVAL] The *trid* argument was not a valid trace stream identifier.

30370 EXAMPLES

30371 None.

30372 APPLICATION USAGE

30373 None.

30374 RATIONALE

30375 None.

30376 FUTURE DIRECTIONS

30377 None.

30378 SEE ALSO

30379 *posix_trace_event()*, *posix_trace_getnext_event()*, *posix_trace_trid_eventid_open()*, the Base
30380 Definitions volume of IEEE Std 1003.1-2001, **<trace.h>**

30381 CHANGE HISTORY

30382 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30383 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

30384 **NAME**30385 posix_trace_flush — trace stream flush from a process (**TRACING**)30386 **SYNOPSIS**30387 TRC `#include <sys/types.h>`30388 `#include <trace.h>`30389 TRC TRL `int posix_trace_flush(trace_id_t trid);` |

30390

30391 **DESCRIPTION**30392 Refer to *posix_trace_create()*.

30393 NAME

30394 `posix_trace_get_attr`, `posix_trace_get_status` — retrieve the trace attributes or trace status
30395 (TRACING)

30396 SYNOPSIS

30397 TRC `#include <trace.h>`

```
30398        int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
30399        int posix_trace_get_status(trace_id_t trid,
30400                                  struct posix_trace_status_info *statusinfo);
30401
```

30402 DESCRIPTION

30403 The `posix_trace_get_attr()` function shall copy the attributes of the active trace stream identified
30404 TRL by *trid* into the object pointed to by the *attr* argument. If the Trace Log option is supported, *trid*
30405 may represent a pre-recorded trace log.

30406 The `posix_trace_get_status()` function shall return, in the structure pointed to by the *statusinfo*
30407 argument, the current trace status for the trace stream identified by the *trid* argument. These
30408 status values returned in the structure pointed to by *statusinfo* shall have been appropriately
30409 TRL read to ensure that the returned values are consistent. If the Trace Log option is supported and
30410 the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the
30411 completed trace stream.

30412 Each time the `posix_trace_get_status()` function is used, the overrun status of the trace stream
30413 TRL shall be reset to `POSIX_TRACE_NO_OVERRUN` immediately after the call completes. If the
30414 Trace Log option is supported, the `posix_trace_get_status()` function shall behave the same as
30415 when the option is not supported except for the following differences:

- 30416 • If the *trid* argument refers to a trace stream with log, each time the `posix_trace_get_status()`
30417 function is used, the log overrun status of the trace stream shall be reset to
30418 `POSIX_TRACE_NO_OVERRUN` and the *flush_error* status shall be reset to zero immediately
30419 after the call completes.
- 30420 • If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the
30421 status of the completed trace stream and the status values of the trace stream shall not be
30422 reset.

30424 RETURN VALUE

30425 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30426 return the corresponding error number.

30427 The `posix_trace_get_attr()` function stores the trace attributes in the object pointed to by *attr*, if
30428 successful.

30429 The `posix_trace_get_status()` function stores the trace status in the object pointed to by *statusinfo*,
30430 if successful.

30431 ERRORS

30432 These functions shall fail if:

30433 [EINVAL] The trace stream argument *trid* does not correspond to a valid active trace
30434 stream or a valid trace log.

30435 **EXAMPLES**

30436 None.

30437 **APPLICATION USAGE**

30438 None.

30439 **RATIONALE**

30440 None.

30441 **FUTURE DIRECTIONS**

30442 None.

30443 **SEE ALSO**30444 *posix_trace_attr_destroy()*, *posix_trace_attr_init()*, *posix_trace_create()*, *posix_trace_open()*, the Base30445 Definitions volume of IEEE Std 1003.1-2001, **<trace.h>**30446 **CHANGE HISTORY**

30447 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30448 IEEE PASC Interpretation 1003.1 #123 is applied.

30449 NAME

30450 posix_trace_get_filter, posix_trace_set_filter — retrieve and set the filter of an initialized trace
30451 stream (**TRACING**)

30452 SYNOPSIS

30453 TRC TEF `#include <trace.h>`

```
30454 int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
30455 int posix_trace_set_filter(trace_id_t trid,
30456     const trace_event_set_t *set, int how);
30457
```

30458 DESCRIPTION

30459 The *posix_trace_get_filter()* function shall retrieve, into the argument pointed to by *set*, the actual
30460 trace event filter from the trace stream specified by *trid*.

30461 The *posix_trace_set_filter()* function shall change the set of filtered trace event types after a trace
30462 stream identified by the *trid* argument is created. This function may be called prior to starting
30463 the trace stream, or while the trace stream is active. By default, if no call is made to
30464 *posix_trace_set_filter()*, all trace events shall be recorded (that is, none of the trace event types are
30465 filtered out).

30466 If this function is called while the trace is in progress, a special system trace event,
30467 POSIX_TRACE_FILTER, shall be recorded in the trace indicating both the old and the new sets
30468 of filtered trace event types (see Table 2-4 (on page 78) and Table 2-6 (on page 79)).

30469 If the *posix_trace_set_filter()* function is interrupted by a signal, an error shall be returned and the
30470 filter shall not be changed. In this case, the state of the trace stream shall not be changed.

30471 The value of the argument *how* indicates the manner in which the set is to be changed and shall
30472 have one of the following values, as defined in the **<trace.h>** header:

30473 POSIX_TRACE_SET_EVENTSET

30474 The resulting set of trace event types to be filtered shall be the trace event type set pointed
30475 to by the argument *set*.

30476 POSIX_TRACE_ADD_EVENTSET

30477 The resulting set of trace event types to be filtered shall be the union of the current set and
30478 the trace event type set pointed to by the argument *set*.

30479 POSIX_TRACE_SUB_EVENTSET

30480 The resulting set of trace event types to be filtered shall be all trace event types in the
30481 current set that are not in the set pointed to by the argument *set*; that is, remove each
30482 element of the specified set from the current filter.

30483 RETURN VALUE

30484 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30485 return the corresponding error number.

30486 The *posix_trace_get_filter()* function stores the set of filtered trace event types in *set*, if successful.

30487 ERRORS

30488 These functions shall fail if:

30489 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream
30490 or the value of the argument pointed to by *set* is invalid.

30491 [EINTR] The operation was interrupted by a signal.

30492 **EXAMPLES**

30493 None.

30494 **APPLICATION USAGE**

30495 None.

30496 **RATIONALE**

30497 None.

30498 **FUTURE DIRECTIONS**

30499 None.

30500 **SEE ALSO**

30501 Table 2-4 (on page 78), Table 2-6 (on page 79), *posix_trace_eventset_add()*, the Base Definitions
30502 volume of IEEE Std 1003.1-2001, <**trace.h**>

30503 **CHANGE HISTORY**

30504 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30505 IEEE PASC Interpretation 1003.1 #123 is applied.

30506 NAME

30507 posix_trace_get_status — retrieve the trace status (**TRACING**)

30508 SYNOPSIS

30509 TRC `#include <trace.h>`

30510 `int posix_trace_get_status(trace_id_t trid,`
30511 `struct posix_trace_status_info *statusinfo);`

30512

30513 DESCRIPTION

30514 Refer to *posix_trace_get_attr()*.

30515 **NAME**

30516 posix_trace_getnext_event, posix_trace_timedgetnext_event, posix_trace_trygetnext_event —
 30517 retrieve a trace event (**TRACING**)

30518 **SYNOPSIS**

```
30519 TRC    #include <sys/types.h>
30520         #include <trace.h>

30521         int posix_trace_getnext_event(trace_id_t trid,
30522         struct posix_trace_event_info *restrict event,
30523         void *restrict data, size_t num_bytes,
30524         size_t *restrict data_len, int *restrict unavailable);
30525 TRC TMO int posix_trace_timedgetnext_event(trace_id_t trid,
30526         struct posix_trace_event_info *restrict event,
30527         void *restrict data, size_t num_bytes,
30528         size_t *restrict data_len, int *restrict unavailable,
30529         const struct timespec *restrict abs_timeout);
30530 TRC    int posix_trace_trygetnext_event(trace_id_t trid,
30531         struct posix_trace_event_info *restrict event,
30532         void *restrict data, size_t num_bytes,
30533         size_t *restrict data_len, int *restrict unavailable);
30534
```

30535 **DESCRIPTION**

30536 The *posix_trace_getnext_event()* function shall report a recorded trace event either from an active
 30537 TRL trace stream without log or a pre-recorded trace stream identified by the *trid* argument. The
 30538 *posix_trace_trygetnext_event()* function shall report a recorded trace event from an active trace
 30539 stream without log identified by the *trid* argument.

30540 The trace event information associated with the recorded trace event shall be copied by the
 30541 function into the structure pointed to by the argument *event* and the data associated with the
 30542 trace event shall be copied into the buffer pointed to by the *data* argument.

30543 The *posix_trace_getnext_event()* function shall block if the *trid* argument identifies an active trace
 30544 stream and there is currently no trace event ready to be retrieved. When returning, if a recorded
 30545 trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero.
 30546 Otherwise, the variable pointed to by the *unavailable* argument shall be set to a value different
 30547 from zero.

30548 TMO The *posix_trace_timedgetnext_event()* function shall attempt to get another trace event from an
 30549 active trace stream without log, as in the *posix_trace_getnext_event()* function. However, if no
 30550 trace event is available from the trace stream, the implied wait shall be terminated when the
 30551 timeout specified by the argument *abs_timeout* expires, and the function shall return the error
 30552 [ETIMEDOUT].

30553 The timeout shall expire when the absolute time specified by *abs_timeout* passes, as measured by
 30554 the clock upon which timeouts are based (that is, when the value of that clock equals or exceeds
 30555 *abs_timeout*), or if the absolute time specified by *abs_timeout* has already passed at the time of the
 30556 call.

30557 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock;
 30558 if the Timers option is not supported, the timeout shall be based on the system clock as returned
 30559 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which
 30560 it is based. The **timespec** data type is defined in the **<time.h>** header.

30561 TMO Under no circumstance shall the function fail with a timeout if a trace event is immediately
 30562 available from the trace stream. The validity of the *abs_timeout* argument need not be checked if

30563	a trace event is immediately available from the trace stream.
30564	The behavior of this function for a pre-recorded trace stream is unspecified.
30565 TRL	The <i>posix_trace_trygetnext_event()</i> function shall not block. This function shall return an error if
30566	the <i>trid</i> argument identifies a pre-recorded trace stream. If a recorded trace event was reported,
30567	the variable pointed to by the <i>unavailable</i> argument shall be set to zero. Otherwise, if no trace
30568	event was reported, the variable pointed to by the <i>unavailable</i> argument shall be set to a value
30569	different from zero.
30570	The argument <i>num_bytes</i> shall be the size of the buffer pointed to by the <i>data</i> argument. The
30571	argument <i>data_len</i> reports to the application the length in bytes of the data record just
30572	transferred. If <i>num_bytes</i> is greater than or equal to the size of the data associated with the trace
30573	event pointed to by the <i>event</i> argument, all the recorded data shall be transferred. In this case, the
30574	<i>truncation-status</i> member of the trace event structure shall be either
30575	POSIX_TRACE_NOT_TRUNCATED, if the trace event data was recorded without truncation
30576	while tracing, or POSIX_TRACE_TRUNCATED_RECORD, if the trace event data was truncated
30577	when it was recorded. If the <i>num_bytes</i> argument is less than the length of recorded trace event
30578	data, the data transferred shall be truncated to a length of <i>num_bytes</i> , the value stored in the
30579	variable pointed to by <i>data_len</i> shall be equal to <i>num_bytes</i> , and the <i>truncation-status</i> member of
30580	the <i>event</i> structure argument shall be set to POSIX_TRACE_TRUNCATED_READ (see the
30581	posix_trace_event_info structure defined in <trace.h>).
30582	The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace
30583	events shall be reported in the order in which they were generated, up to an implementation-
30584	defined time resolution that causes the ordering of trace events occurring very close to each
30585	other to be unknown. Once reported, a trace event cannot be reported again from an active trace
30586	stream. Once a trace event is reported from an active trace stream without log, the trace stream
30587	shall make the resources associated with that trace event available to record future generated
30588	trace events.
30589	RETURN VALUE
30590	Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30591	return the corresponding error number.
30592	If successful, these functions store:
30593	• The recorded trace event in the object pointed to by <i>event</i>
30594	• The trace event information associated with the recorded trace event in the object pointed to
30595	by <i>data</i>
30596	• The length of this trace event information in the object pointed to by <i>data_len</i>
30597	• The value of zero in the object pointed to by <i>unavailable</i>
30598	ERRORS
30599	These functions shall fail if:
30600	[EINVAL] The trace stream identifier argument <i>trid</i> is invalid.
30601	The <i>posix_trace_getnext_event()</i> and <i>posix_trace_timedgetnext_event()</i> functions shall fail if:
30602	[EINTR] The operation was interrupted by a signal, and so the call had no effect.
30603	The <i>posix_trace_trygetnext_event()</i> function shall fail if:
30604	[EINVAL] The trace stream identifier argument <i>trid</i> does not correspond to an active
30605	trace stream.

30606	TMO	The <i>posix_trace_timedgetnext_event()</i> function shall fail if:
30607	[EINVAL]	There is no trace event immediately available from the trace stream, and the
30608		<i>timeout</i> argument is invalid.
30609	[ETIMEDOUT]	No trace event was available from the trace stream before the specified
30610		timeout <i>timeout</i> expired.
30611		

30612 **EXAMPLES**

30613 None.

30614 **APPLICATION USAGE**

30615 None.

30616 **RATIONALE**

30617 None.

30618 **FUTURE DIRECTIONS**

30619 None.

30620 **SEE ALSO**

30621 *posix_trace_create()*, *posix_trace_open()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 30622 **<sys/types.h>**, **<trace.h>**

30623 **CHANGE HISTORY**

30624 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30625 IEEE PASC Interpretation 1003.1 #123 is applied.

30626 NAME

30627 posix_trace_open, posix_trace_rewind — trace log management (**TRACING**)

30628 SYNOPSIS

30629 TRC TRL `#include <trace.h>`

30630 `int posix_trace_open(int file_desc, trace_id_t *trid);`

30631 `int posix_trace_rewind(trace_id_t trid);`

30632

30633 DESCRIPTION

30634 Refer to *posix_trace_close()*.

30635 **NAME**30636 posix_trace_set_filter — set filter of an initialized trace stream (**TRACING**)30637 **SYNOPSIS**30638 TRC TEF `#include <trace.h>`30639 int posix_trace_set_filter(trace_id_t trid,
30640 const trace_event_set_t *set, int how);

30641

30642 **DESCRIPTION**30643 Refer to *posix_trace_get_filter()*.

30644 NAME

30645 posix_trace_shutdown — trace stream shutdown from a process (**TRACING**)

30646 SYNOPSIS

30647 TRC #include <sys/types.h>

30648 #include <trace.h>

30649 int posix_trace_shutdown(trace_id_t *trid*);

30650

30651 DESCRIPTION

30652 Refer to *posix_trace_create()*.

30653 **NAME**30654 posix_trace_start, posix_trace_stop — trace start and stop (**TRACING**)30655 **SYNOPSIS**

30656 TRC #include <trace.h>

30657 int posix_trace_start(trace_id_t trid);

30658 int posix_trace_stop (trace_id_t trid);

30659

30660 **DESCRIPTION**30661 The *posix_trace_start()* and *posix_trace_stop()* functions, respectively, shall start and stop the
30662 trace stream identified by the argument *trid*.30663 The effect of calling the *posix_trace_start()* function shall be recorded in the trace stream as the
30664 POSIX_TRACE_START system trace event and the status of the trace stream shall become
30665 POSIX_TRACE_RUNNING. If the trace stream is in progress when this function is called, the
30666 POSIX_TRACE_START system trace event shall not be recorded and the trace stream shall
30667 continue to run. If the trace stream is full, the POSIX_TRACE_START system trace event shall
30668 not be recorded and the status of the trace stream shall not be changed.30669 The effect of calling the *posix_trace_stop()* function shall be recorded in the trace stream as the
30670 POSIX_TRACE_STOP system trace event and the status of the trace stream shall become
30671 POSIX_TRACE_SUSPENDED. If the trace stream is suspended when this function is called, the
30672 POSIX_TRACE_STOP system trace event shall not be recorded and the trace stream shall remain
30673 suspended. If the trace stream is full, the POSIX_TRACE_STOP system trace event shall not be
30674 recorded and the status of the trace stream shall not be changed.30675 **RETURN VALUE**30676 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
30677 return the corresponding error number.30678 **ERRORS**

30679 These functions shall fail if:

30680 [EINVAL] The value of the argument *trid* does not correspond to an active trace stream
30681 and thus no trace stream was started or stopped.30682 [EINTR] The operation was interrupted by a signal and thus the trace stream was not
30683 necessarily started or stopped.30684 **EXAMPLES**

30685 None.

30686 **APPLICATION USAGE**

30687 None.

30688 **RATIONALE**

30689 None.

30690 **FUTURE DIRECTIONS**

30691 None.

30692 **SEE ALSO**30693 *posix_trace_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <trace.h>

30694 **CHANGE HISTORY**

30695 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30696 IEEE PASC Interpretation 1003.1 #123 is applied.

30697 **NAME**30698 `posix_trace_timedgetnext_event`, — retrieve a trace event (**TRACING**)30699 **SYNOPSIS**30700 TRC TMO `#include <sys/types.h>`30701 `#include <trace.h>`

```
30702     int posix_trace_timedgetnext_event(trace_id_t trid,  
30703         struct posix_trace_event_info *restrict event,  
30704         void *restrict data, size_t num_bytes,  
30705         size_t *restrict data_len, int *restrict unavailable,  
30706         const struct timespec *restrict abs_timeout);  
30707
```

30708 **DESCRIPTION**30709 Refer to `posix_trace_getnext_event()`.

30710 NAME

30711 posix_trace_trid_eventid_open — open a trace event type identifier (**TRACING**)

30712 SYNOPSIS

30713 TRC TEF `#include <trace.h>`

```
30714        int posix_trace_trid_eventid_open(trace_id_t trid,  
30715                                          const char *restrict event_name,  
30716                                          trace_event_id_t *restrict event);  
30717
```

30718 DESCRIPTION

30719 Refer to *posix_trace_eventid_equal()*.

30720 **NAME**30721 `posix_trace_trygetnext_event` — retrieve a trace event (**TRACING**)30722 **SYNOPSIS**30723 TRC `#include <sys/types.h>`30724 `#include <trace.h>`

```
30725        int posix_trace_trygetnext_event(trace_id_t trid,  
30726                    struct posix_trace_event_info *restrict event,  
30727                    void *restrict data, size_t num_bytes,  
30728                    size_t *restrict data_len, int *restrict unavailable);
```

30729

30730 **DESCRIPTION**30731 Refer to `posix_trace_getnext_event()`.

30732 NAME

30733 posix_typed_mem_get_info — query typed memory information (**ADVANCED REALTIME**)

30734 SYNOPSIS

30735 **TYM** #include <sys/mman.h>

```
30736 int posix_typed_mem_get_info(int fildes,
30737 struct posix_typed_mem_info *info);
```

30738

30739 DESCRIPTION

30740 The *posix_typed_mem_get_info()* function shall return, in the *posix_tmi_length* field of the
 30741 **posix_typed_mem_info** structure pointed to by *info*, the maximum length which may be
 30742 successfully allocated by the typed memory object designated by *fildes*. This maximum length
 30743 shall take into account the flag **POSIX_TYPED_MEM_ALLOCATE** or
 30744 **POSIX_TYPED_MEM_ALLOCATE_CONTIG** specified when the typed memory object
 30745 represented by *fildes* was opened. The maximum length is dynamic; therefore, the value returned
 30746 is valid only while the current mapping of the corresponding typed memory pool remains
 30747 unchanged.

30748 If *fildes* represents a typed memory object opened with neither the
 30749 **POSIX_TYPED_MEM_ALLOCATE** flag nor the **POSIX_TYPED_MEM_ALLOCATE_CONTIG**
 30750 flag specified, the returned value of *info*->*posix_tmi_length* is unspecified.

30751 The *posix_typed_mem_get_info()* function may return additional implementation-defined
 30752 information in other fields of the **posix_typed_mem_info** structure pointed to by *info*.

30753 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this
 30754 function is undefined.

30755 RETURN VALUE

30756 Upon successful completion, the *posix_typed_mem_get_info()* function shall return zero;
 30757 otherwise, the corresponding error status value shall be returned.

30758 ERRORS

30759 The *posix_typed_mem_get_info()* function shall fail if:

30760 [EBADF] The *fildes* argument is not a valid open file descriptor.

30761 [ENODEV] The *fildes* argument is not connected to a memory object supported by this
 30762 function.

30763 This function shall not return an error code of [EINTR].

30764 EXAMPLES

30765 None.

30766 APPLICATION USAGE

30767 None.

30768 RATIONALE

30769 An application that needs to allocate a block of typed memory with length dependent upon the
 30770 amount of memory currently available must either query the typed memory object to obtain the
 30771 amount available, or repeatedly invoke *mmap()* attempting to guess an appropriate length.
 30772 While the latter method is existing practice with *malloc()*, it is awkward and imprecise. The
 30773 *posix_typed_mem_get_info()* function allows an application to immediately determine available
 30774 memory. This is particularly important for typed memory objects that may in some cases be
 30775 scarce resources. Note that when a typed memory pool is a shared resource, some form of
 30776 mutual-exclusion or synchronization may be required while typed memory is being queried and

30777 allocated to prevent race conditions.

30778 The existing *fstat()* function is not suitable for this purpose. We realize that implementations
30779 may wish to provide other attributes of typed memory objects (for example, alignment
30780 requirements, page size, and so on). The *fstat()* function returns a structure which is not
30781 extensible and, furthermore, contains substantial information that is inappropriate for typed
30782 memory objects.

30783 **FUTURE DIRECTIONS**

30784 None.

30785 **SEE ALSO**

30786 *fstat()*, *mmap()*, *posix_typed_mem_open()*, the Base Definitions volume of IEEE Std 1003.1-2001,
30787 **<sys/mman.h>**

30788 **CHANGE HISTORY**

30789 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

30790 NAME

30791 posix_typed_mem_open — open a typed memory object (**ADVANCED REALTIME**)

30792 SYNOPSIS

30793 TYM `#include <sys/mman.h>`

30794 `int posix_typed_mem_open(const char *name, int oflag, int tflag);`
 30795

30796 DESCRIPTION

30797 The *posix_typed_mem_open()* function shall establish a connection between the typed memory
 30798 object specified by the string pointed to by *name* and a file descriptor. It shall create an open file
 30799 description that refers to the typed memory object and a file descriptor that refers to that open
 30800 file description. The file descriptor is used by other functions to refer to that typed memory
 30801 object. It is unspecified whether the name appears in the file system and is visible to other
 30802 functions that take pathnames as arguments. The *name* argument shall conform to the
 30803 construction rules for a pathname. If *name* begins with the slash character, then processes calling
 30804 *posix_typed_mem_open()* with the same value of *name* shall refer to the same typed memory
 30805 object. If *name* does not begin with the slash character, the effect is implementation-defined. The
 30806 interpretation of slash characters other than the leading slash character in *name* is
 30807 implementation-defined.

30808 Each typed memory object supported in a system shall be identified by a name which specifies
 30809 not only its associated typed memory pool, but also the path or port by which it is accessed. That
 30810 is, the same typed memory pool accessed via several different ports shall have several different
 30811 corresponding names. The binding between names and typed memory objects is established in
 30812 an implementation-defined manner. Unlike shared memory objects, there is no way within
 30813 IEEE Std 1003.1-2001 for a program to create a typed memory object.

30814 The value of *tflag* shall determine how the typed memory object behaves when subsequently
 30815 mapped by calls to *mmap()*. At most, one of the following flags defined in **<sys/mman.h>** may
 30816 be specified:

30817 POSIX_TYPED_MEM_ALLOCATE

30818 Allocate on *mmap()*.

30819 POSIX_TYPED_MEM_ALLOCATE_CONTIG

30820 Allocate contiguously on *mmap()*.

30821 POSIX_TYPED_MEM_MAP_ALLOCATABLE

30822 Map on *mmap()*, without affecting allocatability.

30823 If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE specified, any subsequent call to *mmap()*
 30824 using the returned file descriptor shall result in allocation and mapping of typed memory from
 30825 the specified typed memory pool. The allocated memory may be a contiguous previously
 30826 unallocated area of the typed memory pool or several non-contiguous previously unallocated
 30827 areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag
 30828 POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap()* using the
 30829 returned file descriptor shall result in allocation and mapping of a single contiguous previously
 30830 unallocated area of the typed memory pool (also mapped to a contiguous portion of the process
 30831 address space). If *tflag* has none of the flags POSIX_TYPED_MEM_ALLOCATE or
 30832 POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap()* using the
 30833 returned file descriptor shall map an application-chosen area from the specified typed memory
 30834 pool such that this mapped area becomes unavailable for allocation until unmapped by all
 30835 processes. If *tflag* has the flag POSIX_TYPED_MEM_MAP_ALLOCATABLE specified, any
 30836 subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen
 30837 area from the specified typed memory pool without an effect on the availability of that area for

allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. The appropriate privilege to specify the `POSIX_TYPED_MEM_MAP_ALLOCATABLE` flag is implementation-defined.

If successful, `posix_typed_mem_open()` shall return a file descriptor for the typed memory object that is the lowest numbered file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor shall not share it with any other processes. It is unspecified whether the file offset is set. The `FD_CLOEXEC` file descriptor flag associated with the new file descriptor shall be cleared.

The behavior of `msync()`, `ftruncate()`, and all file operations other than `mmap()`, `posix_mem_offset()`, `posix_typed_mem_get_info()`, `fstat()`, `dup()`, `dup2()`, and `close()`, is unspecified when passed a file descriptor connected to a typed memory object by this function.

The file status flags of the open file description shall be set according to the value of *oflag*. Applications shall specify exactly one of the three access mode values described below and defined in the `<fcntl.h>` header, as the value of *oflag*.

30853	<code>O_RDONLY</code>	Open for read access only.
30854	<code>O_WRONLY</code>	Open for write access only.
30855	<code>O_RDWR</code>	Open for read or write access.

30856 RETURN VALUE

Upon successful completion, the `posix_typed_mem_open()` function shall return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, it shall return `-1` and set *errno* to indicate the error.

30860 ERRORS

The `posix_typed_mem_open()` function shall fail if:

30862 30863	<code>[EACCES]</code>	The typed memory object exists and the permissions specified by <i>oflag</i> are denied.
30864	<code>[EINTR]</code>	The <code>posix_typed_mem_open()</code> operation was interrupted by a signal.
30865 30866 30867 30868	<code>[EINVAL]</code>	The flags specified in <i>tflag</i> are invalid (more than one of <code>POSIX_TYPED_MEM_ALLOCATE</code> , <code>POSIX_TYPED_MEM_ALLOCATE_CONTIG</code> , or <code>POSIX_TYPED_MEM_MAP_ALLOCATABLE</code> is specified).
30869	<code>[EMFILE]</code>	Too many file descriptors are currently in use by this process.
30870 30871 30872	<code>[ENAMETOOLONG]</code>	The length of the <i>name</i> argument exceeds <code>{PATH_MAX}</code> or a pathname component is longer than <code>{NAME_MAX}</code> .
30873	<code>[ENFILE]</code>	Too many file descriptors are currently open in the system.
30874	<code>[ENOENT]</code>	The named typed memory object does not exist.
30875 30876	<code>[EPERM]</code>	The caller lacks the appropriate privilege to specify the flag <code>POSIX_TYPED_MEM_MAP_ALLOCATABLE</code> in argument <i>tflag</i> .

30877 EXAMPLES

30878 None.

30879 APPLICATION USAGE

30880 None.

30881 RATIONALE

30882 None.

30883 FUTURE DIRECTIONS

30884 None.

30885 SEE ALSO

30886 *close()*, *dup()*, *exec*, *fcntl()*, *fstat()*, *ftruncate()*, *mmap()*, *msync()*, *posix_mem_offset()*,
30887 *posix_typed_mem_get_info()*, *umask()*, the Base Definitions volume of IEEE Std 1003.1-2001,
30888 <fcntl.h>, <sys/mman.h>

30889 CHANGE HISTORY

30890 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

30891 **NAME**

30892 pow, powf, powl — power function

30893 **SYNOPSIS**

30894 #include <math.h>

30895 double pow(double x, double y);

30896 float powf(float x, float y);

30897 long double powl(long double x, long double y);

30898 **DESCRIPTION**

30899 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 30900 conflict between the requirements described here and the ISO C standard is unintentional. This
 30901 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

30902 These functions shall compute the value of x raised to the power y , x^y . If x is negative, the
 30903 application shall ensure that y is an integer value.

30904 An application wishing to check for error situations should set *errno* to zero and call
 30905 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 30906 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 30907 zero, an error has occurred.

30908 **RETURN VALUE**30909 Upon successful completion, these functions shall return the value of x raised to the power y .

30910 **MX** For finite values of $x < 0$, and finite non-integer values of y , a domain error shall occur and either
 30911 a NaN (if representable), or an implementation-defined value shall be returned.

30912 If the correct value would cause overflow, a range error shall occur and *pow()*, *powf()*, and
 30913 *powl()* shall return $\pm\text{HUGE\_VAL}$, $\pm\text{HUGE\_VALF}$, and $\pm\text{HUGE\_VALL}$, respectively, with the
 30914 same sign as the correct value of the function.

30915 If the correct value would cause underflow, and is not representable, a range error may occur,
 30916 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

30917 **MX** If x or y is a NaN, a NaN shall be returned (unless specified elsewhere in this description).30918 For any value of y (including NaN), if x is +1, 1.0 shall be returned.30919 For any value of x (including NaN), if y is ± 0 , 1.0 shall be returned.30920 For any odd integer value of $y > 0$, if x is ± 0 , ± 0 shall be returned.30921 For $y > 0$ and not an odd integer, if x is ± 0 , +0 shall be returned.30922 If x is -1 , and y is $\pm\text{Inf}$, 1.0 shall be returned.30923 For $|x| < 1$, if y is $-\text{Inf}$, $+\text{Inf}$ shall be returned.30924 For $|x| > 1$, if y is $-\text{Inf}$, +0 shall be returned.30925 For $|x| < 1$, if y is $+\text{Inf}$, +0 shall be returned.30926 For $|x| > 1$, if y is $+\text{Inf}$, $+\text{Inf}$ shall be returned.30927 For y an odd integer < 0 , if x is $-\text{Inf}$, -0 shall be returned.30928 For $y < 0$ and not an odd integer, if x is $-\text{Inf}$, +0 shall be returned.30929 For y an odd integer > 0 , if x is $-\text{Inf}$, $-\text{Inf}$ shall be returned.30930 For $y > 0$ and not an odd integer, if x is $-\text{Inf}$, $+\text{Inf}$ shall be returned.

30931 For $y < 0$, if x is $+\text{Inf}$, $+0$ shall be returned.

30932 For $y > 0$, if x is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

30933 For y an odd integer < 0 , if x is ± 0 , a pole error shall occur and $\pm\text{HUGE\_VAL}$, $\pm\text{HUGE\_VALF}$, and
 30934 $\pm\text{HUGE\_VALL}$ shall be returned for $\text{pow}()$, $\text{powf}()$, and $\text{powl}()$, respectively.

30935 For $y < 0$ and not an odd integer, if x is ± 0 , a pole error shall occur and HUGE_VAL ,
 30936 HUGE_VALF , and HUGE_VALL shall be returned for $\text{pow}()$, $\text{powf}()$, and $\text{powl}()$, respectively.

30937 If the correct value would cause underflow, and is representable, a range error may occur and
 30938 the correct value shall be returned.

30939 ERRORS

30940 These functions shall fail if:

30941 Domain Error The value of x is negative and y is a finite non-integer.

30942 If the integer expression $(\text{math\_errhandling} \ \& \ \text{MATH\_ERRNO})$ is non-zero,
 30943 then *errno* shall be set to $[\text{EDOM}]$. If the integer expression $(\text{math\_errhandling}$
 30944 $\ \& \ \text{MATH\_ERREXCEPT})$ is non-zero, then the invalid floating-point exception
 30945 shall be raised.

30946 MX Pole Error The value of x is zero and y is negative.

30947 If the integer expression $(\text{math\_errhandling} \ \& \ \text{MATH\_ERRNO})$ is non-zero,
 30948 then *errno* shall be set to $[\text{ERANGE}]$. If the integer expression
 30949 $(\text{math\_errhandling} \ \& \ \text{MATH\_ERREXCEPT})$ is non-zero, then the divide-by-
 30950 zero floating-point exception shall be raised.

30951 Range Error The result overflows.

30952 If the integer expression $(\text{math\_errhandling} \ \& \ \text{MATH\_ERRNO})$ is non-zero,
 30953 then *errno* shall be set to $[\text{ERANGE}]$. If the integer expression
 30954 $(\text{math\_errhandling} \ \& \ \text{MATH\_ERREXCEPT})$ is non-zero, then the overflow
 30955 floating-point exception shall be raised.

30956 These functions may fail if:

30957 Range Error The result underflows.

30958 If the integer expression $(\text{math\_errhandling} \ \& \ \text{MATH\_ERRNO})$ is non-zero,
 30959 then *errno* shall be set to $[\text{ERANGE}]$. If the integer expression
 30960 $(\text{math\_errhandling} \ \& \ \text{MATH\_ERREXCEPT})$ is non-zero, then the underflow
 30961 floating-point exception shall be raised.

30962 EXAMPLES

30963 None.

30964 APPLICATION USAGE

30965 On error, the expressions $(\text{math\_errhandling} \ \& \ \text{MATH\_ERRNO})$ and $(\text{math\_errhandling} \ \& \ \text{MATH\_ERREXCEPT})$
 30966 are independent of each other, but at least one of them must be non-zero.

30967 RATIONALE

30968 None.

30969 FUTURE DIRECTIONS

30970 None.

30971 **SEE ALSO**

30972 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
30973 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

30974 **CHANGE HISTORY**

30975 First released in Issue 1. Derived from Issue 1 of the SVID.

30976 **Issue 5**

30977 The DESCRIPTION is updated to indicate how an application should check for an error. This
30978 text was previously published in the APPLICATION USAGE section.

30979 **Issue 6**

30980 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

30981 The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

30982 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
30983 revised to align with the ISO/IEC 9899:1999 standard.

30984 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
30985 marked.

30986 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/42 is applied, correcting the third
30987 paragraph in the RETURN VALUE section.

30988 **NAME**

30989

```
pread
```

 — read from a file

30990 **SYNOPSIS**

30991 xSI

```
#include <unistd.h>
```

30992

```
ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
```

30993

30994 **DESCRIPTION**

30995 Refer to *read()*.

30996 **NAME**

30997 printf — print formatted output

30998 **SYNOPSIS**

30999 #include <stdio.h>

31000 int printf(const char *restrict *format*, ...);31001 **DESCRIPTION**31002 Refer to *fprintf()*.

31003 **NAME**

31004 pselect, select — synchronous I/O multiplexing

31005 **SYNOPSIS**

```

31006     #include <sys/select.h>

31007     int pselect(int nfds, fd_set *restrict readfds,
31008                 fd_set *restrict writefds, fd_set *restrict errorfds,
31009                 const struct timespec *restrict timeout,
31010                 const sigset_t *restrict sigmask);
31011     int select(int nfds, fd_set *restrict readfds,
31012                fd_set *restrict writefds, fd_set *restrict errorfds,
31013                struct timeval *restrict timeout);
31014     void FD_CLR(int fd, fd_set *fdset);
31015     int FD_ISSET(int fd, fd_set *fdset);
31016     void FD_SET(int fd, fd_set *fdset);
31017     void FD_ZERO(fd_set *fdset);

```

31018 **DESCRIPTION**

31019 The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the
 31020 *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for
 31021 reading, are ready for writing, or have an exceptional condition pending, respectively.

31022 The *select()* function shall be equivalent to the *pselect()* function, except as follows:

- 31023 • For the *select()* function, the timeout period is given in seconds and microseconds in an
 31024 argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is
 31025 given in seconds and nanoseconds in an argument of type **struct timespec**.
- 31026 • The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when *sigmask*
 31027 is a null pointer.
- 31028 • Upon successful completion, the *select()* function may modify the object pointed to by the
 31029 *timeout* argument.

31030 The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal
 31031 XSR devices, **STREAMS**-based files, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()*
 31032 on file descriptors that refer to other types of file is unspecified.

31033 The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall
 31034 be checked in each set; that is, the descriptors from zero through *nfds*–1 in the descriptor sets
 31035 shall be examined.

31036 If the *readfds* argument is not a null pointer, it points to an object of type **fd_set** that on input
 31037 specifies the file descriptors to be checked for being ready to read, and on output indicates
 31038 which file descriptors are ready to read.

31039 If the *writefds* argument is not a null pointer, it points to an object of type **fd_set** that on input
 31040 specifies the file descriptors to be checked for being ready to write, and on output indicates
 31041 which file descriptors are ready to write.

31042 If the *errorfds* argument is not a null pointer, it points to an object of type **fd_set** that on input
 31043 specifies the file descriptors to be checked for error conditions pending, and on output indicates
 31044 which file descriptors have error conditions pending.

31045 Upon successful completion, the *pselect()* or *select()* function shall modify the objects pointed to
 31046 by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for
 31047 reading, ready for writing, or have an error condition pending, respectively, and shall return the
 31048 total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the

31049 corresponding bit shall be set on successful completion if it was set on input and the associated
31050 condition is true for that file descriptor.

31051 If none of the selected descriptors are ready for the requested operation, the *pselect()* or *select()*
31052 function shall block until at least one of the requested operations becomes ready, until the
31053 *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the
31054 *pselect()* or *select()* function shall take before timing out. If the *timeout* parameter is not a null
31055 pointer, it specifies a maximum interval to wait for the selection to complete. If the specified
31056 time interval expires without any requested operation becoming ready, the function shall return.
31057 If the *timeout* parameter is a null pointer, then the call to *pselect()* or *select()* shall block
31058 indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout*
31059 parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

31060 The use of a timeout does not affect any pending timers set up by *alarm()*, *ualarm()*, or
31061 *setitimer()*.

31062 Implementations may place limitations on the maximum timeout interval supported. All
31063 implementations shall support a maximum timeout interval of at least 31 days. If the *timeout*
31064 argument specifies a timeout interval greater than the implementation-defined maximum value,
31065 the maximum value shall be used as the actual timeout value. Implementations may also place
31066 limitations on the granularity of timeout intervals. If the requested timeout interval requires a
31067 finer granularity than the implementation supports, the actual timeout interval shall be rounded
31068 up to the next supported value.

31069 If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the
31070 process by the set of signals pointed to by *sigmask* before examining the descriptors, and shall
31071 restore the signal mask of the process before returning.

31072 A descriptor shall be considered ready for reading when a call to an input function with
31073 O_NONBLOCK clear would not block, whether or not the function would transfer data
31074 successfully. (The function might return data, an end-of-file indication, or an error other than
31075 one indicating that it is blocked, and in each of these cases the descriptor shall be considered
31076 ready for reading.)

31077 A descriptor shall be considered ready for writing when a call to an output function with
31078 O_NONBLOCK clear would not block, whether or not the function would transfer data
31079 successfully.

31080 If a socket has a pending error, it shall be considered to have an exceptional condition pending.
31081 Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for
31082 use with a socket, it is protocol-specific except as noted below. For other file types it is
31083 implementation-defined. If the operation is meaningless for a particular file type, *pselect()* or
31084 *select()* shall indicate that the descriptor is ready for read or write operations, and shall indicate
31085 that the descriptor has no exceptional condition pending.

31086 If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with
31087 parameters requesting normal and ancillary data, such that the presence of either type shall
31088 cause the socket to be marked as readable. The presence of out-of-band data shall be checked if
31089 the socket option SO_OOBINLINE has been enabled, as out-of-band data is enqueued with
31090 normal data. If the socket is currently listening, then it shall be marked as readable if an
31091 incoming connection request has been received, and a call to the *accept()* function shall complete
31092 without blocking.

31093 If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying
31094 an amount of normal data equal to the current value of the SO_SNDLOWAT option for the
31095 socket. If a non-blocking call to the *connect()* function has been made for a socket, and the
31096 connection attempt has either succeeded or failed leaving a pending error, the socket shall be

31097 marked as writable.

31098 A socket shall be considered to have an exceptional condition pending if a receive operation
 31099 with `O_NONBLOCK` clear for the open file description and with the `MSG_OOB` flag set would
 31100 return out-of-band data without blocking. (It is protocol-specific whether the `MSG_OOB` flag
 31101 would be used to read out-of-band data.) A socket shall also be considered to have an
 31102 exceptional condition pending if an out-of-band data mark is present in the receive queue. Other
 31103 circumstances under which a socket may be considered to have an exceptional condition
 31104 pending are protocol-specific and implementation-defined.

31105 If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is not
 31106 a null pointer, the *pselect()* or *select()* function shall block for the time specified, or until
 31107 interrupted by a signal. If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the
 31108 *timeout* argument is a null pointer, the *pselect()* or *select()* function shall block until interrupted
 31109 by a signal.

31110 File descriptors associated with regular files shall always select true for ready to read, ready to
 31111 write, and error conditions.

31112 On failure, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall not be
 31113 modified. If the timeout interval expires without the specified condition being true for any of the
 31114 specified file descriptors, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments
 31115 shall have all bits set to 0.

31116 File descriptor masks of type **fd_set** can be initialized and tested with *FD_CLR()*, *FD_ISSET()*,
 31117 *FD_SET()*, and *FD_ZERO()*. It is unspecified whether each of these is a macro or a function. If a
 31118 macro definition is suppressed in order to access an actual function, or a program defines an
 31119 external identifier with any of these names, the behavior is undefined.

31120 *FD_CLR(fd, fdsetp)* shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not
 31121 a member of this set, there shall be no effect on the set, nor will an error be returned.

31122 *FD_ISSET(fd, fdsetp)* shall evaluate to non-zero if the file descriptor *fd* is a member of the set
 31123 pointed to by *fdsetp*, and shall evaluate to zero otherwise.

31124 *FD_SET(fd, fdsetp)* shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file
 31125 descriptor *fd* is already in this set, there shall be no effect on the set, nor will an error be returned.

31126 *FD_ZERO(fdsetp)* shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is
 31127 returned if the set is not empty at the time *FD_ZERO()* is invoked.

31128 The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or
 31129 equal to `FD_SETSIZE`, or if *fd* is not a valid file descriptor, or if any of the arguments are
 31130 expressions with side effects.

31131 **RETURN VALUE**

31132 Upon successful completion, the *pselect()* and *select()* functions shall return the total number of
 31133 bits set in the bit masks. Otherwise, `-1` shall be returned, and *errno* shall be set to indicate the
 31134 error.

31135 *FD_CLR()*, *FD_SET()*, and *FD_ZERO()* do not return a value. *FD_ISSET()* shall return a non-
 31136 zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and
 31137 0 otherwise.

31138 **ERRORS**

31139 Under the following conditions, *pselect()* and *select()* shall fail and set *errno* to:

31140 [EBADF] One or more of the file descriptor sets specified a file descriptor that is not a
 31141 valid open file descriptor.

31142	[EINTR]	The function was interrupted before any of the selected events occurred and
31143		before the timeout interval expired.
31144 XSI		If SA_RESTART has been set for the interrupting signal, it is implementation-
31145		defined whether the function restarts or returns with [EINTR].
31146	[EINVAL]	An invalid timeout interval was specified.
31147	[EINVAL]	The <i>nfds</i> argument is less than 0 or greater than FD_SETSIZE.
31148 XSR	[EINVAL]	One of the specified file descriptors refers to a STREAM or multiplexer that is
31149		linked (directly or indirectly) downstream from a multiplexer.

31150 EXAMPLES

31151 None.

31152 APPLICATION USAGE

31153 None.

31154 RATIONALE

31155 In previous versions of the Single UNIX Specification, the *select()* function was defined in the
 31156 **<sys/time.h>** header. This is now changed to **<sys/select.h>**. The rationale for this change was
 31157 as follows: the introduction of the *pselect()* function included the **<sys/select.h>** header and the
 31158 **<sys/select.h>** header defines all the related definitions for the *pselect()* and *select()* functions.
 31159 Backwards-compatibility to existing XSI implementations is handled by allowing **<sys/time.h>**
 31160 to include **<sys/select.h>**.

31161 FUTURE DIRECTIONS

31162 None.

31163 SEE ALSO

31164 *accept()*, *alarm()*, *connect()*, *fcntl()*, *poll()*, *read()*, *recvmsg()*, *sendmsg()*, *setitimer()*, *ualarm()*,
 31165 *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/select.h>**, **<sys/time.h>**

31166 CHANGE HISTORY

31167 First released in Issue 4, Version 2.

31168 Issue 5

31169 Moved from X/OPEN UNIX extension to BASE.

31170 In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when
 31171 *nfds* is less than 0 or greater than FD_SETSIZE. It previously stated less than 0, or greater than or
 31172 equal to FD_SETSIZE.

31173 Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.

31174 Issue 6

31175 The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs*
 31176 in the *select()* DESCRIPTION to be *readfds* and *writefds*.

31177 Text referring to sockets is added to the DESCRIPTION.

31178 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are
 31179 marked as part of the XSI STREAMS Option Group.

31180 The following new requirements on POSIX implementations derive from alignment with the
 31181 Single UNIX Specification:

- 31182 • These functions are now mandatory.

31183 The *pselect()* function is added for alignment with IEEE Std 1003.1g-2000 and additional detail
 31184 related to sockets semantics is added to the DESCRIPTION.

31185 The *select()* function now requires inclusion of **<sys/select.h>**.

31186 The **restrict** keyword is added to the *select()* prototype for alignment with the

31187 ISO/IEC 9899:1999 standard.

31188 **NAME**

31189 pthread_atfork — register fork handlers

31190 **SYNOPSIS**

31191 THR #include <pthread.h>

```
31192 int pthread_atfork(void (*prepare)(void), void (*parent)(void),
31193 void (*child)(void));
```

31194

31195 **DESCRIPTION**

31196 The *pthread_atfork()* function shall declare fork handlers to be called before and after *fork()*, in
 31197 the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()*
 31198 processing commences. The *parent* fork handler shall be called after *fork()* processing completes
 31199 in the parent process. The *child* fork handler shall be called after *fork()* processing completes
 31200 in the child process. If no handling is desired at one or more of these three points, the
 31201 corresponding fork handler address(es) may be set to NULL.

31202 The order of calls to *pthread_atfork()* is significant. The *parent* and *child* fork handlers shall be
 31203 called in the order in which they were established by calls to *pthread_atfork()*. The *prepare* fork
 31204 handlers shall be called in the opposite order.

31205 **RETURN VALUE**

31206 Upon successful completion, *pthread_atfork()* shall return a value of zero; otherwise, an error
 31207 number shall be returned to indicate the error.

31208 **ERRORS**31209 The *pthread_atfork()* function shall fail if:

31210 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

31211 The *pthread_atfork()* function shall not return an error code of [EINTR].31212 **EXAMPLES**

31213 None.

31214 **APPLICATION USAGE**

31215 None.

31216 **RATIONALE**

31217 There are at least two serious problems with the semantics of *fork()* in a multi-threaded
 31218 program. One problem has to do with state (for example, memory) covered by mutexes.
 31219 Consider the case where one thread has a mutex locked and the state covered by that mutex is
 31220 inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state
 31221 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply
 31222 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about
 31223 how to correct or otherwise deal with the inconsistent state in the child.

31224 It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the
 31225 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe
 31226 library routines are promised to be available.

31227 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application
 31228 programs may not be aware that a multi-threaded library is in use, and they feel free to call any
 31229 number of library routines between the *fork()* and *exec* calls, just as they always have. Indeed,
 31230 they may be extant single-threaded programs and cannot, therefore, be expected to obey new
 31231 restrictions imposed by the threads library.

On the other hand, the multi-threaded library needs a way to protect its internal state during *fork()* in case it is re-entered later in the child process. The problem arises especially in multi-threaded I/O libraries, which are almost sure to be invoked between the *fork()* and *exec* calls to effect I/O redirection. The solution may require locking mutex variables during *fork()*, or it may entail simply resetting the state in the child after the *fork()* processing completes.

The *pthread_atfork()* function provides multi-threaded libraries with a means to protect themselves from innocent application programs that call *fork()*, and it provides multi-threaded application programs with a standard mechanism for protecting themselves from *fork()* calls in a library routine or the application itself.

The expected usage is that the *prepare* handler acquires all mutex locks and the other two fork handlers release them.

For example, an application can supply a *prepare* routine that acquires the necessary mutexes the library maintains and supply *child* and *parent* routines that release those mutexes, thus ensuring that the child gets a consistent snapshot of the state of the library (and that no mutexes are left stranded). Alternatively, some libraries might be able to supply just a *child* routine that reinitializes the mutexes in the library and all associated states to some known value (for example, what it was when the image was originally executed).

When *fork()* is called, only the calling thread is duplicated in the child process. Synchronization variables remain in the same state in the child as they were in the parent at the time *fork()* was called. Thus, for example, mutex locks may be held by threads that no longer exist in the child process, and any associated states may be inconsistent. The parent process may avoid this by explicit code that acquires and releases locks critical to the child via *pthread_atfork()*. In addition, any critical threads need to be recreated and reinitialized to the proper state in the child (also via *pthread_atfork()*).

A higher-level package may acquire locks on its own data structures before invoking lower-level packages. Under this scenario, the order specified for fork handler calls allows a simple rule of initialization for avoiding package deadlock: a package initializes all packages on which it depends before it calls the *pthread_atfork()* function for itself.

31260 FUTURE DIRECTIONS

31261 None.

31262 SEE ALSO

31263 *atexit()*, *fork()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>

31264 CHANGE HISTORY

31265 First released in Issue 5. Derived from the POSIX Threads Extension.

31266 IEEE PASC Interpretation 1003.1c #4 is applied.

31267 Issue 6

31268 The *pthread_atfork()* function is marked as part of the Threads option.

31269 The <pthread.h> header is added to the SYNOPSIS.

31270 NAME

31271 pthread_attr_destroy, pthread_attr_init — destroy and initialize the thread attributes object

31272 SYNOPSIS

31273 THR `#include <pthread.h>`

31274 `int pthread_attr_destroy(pthread_attr_t *attr);`

31275 `int pthread_attr_init(pthread_attr_t *attr);`

31276

31277 DESCRIPTION

31278 The *pthread_attr_destroy()* function shall destroy a thread attributes object. An implementation
 31279 may cause *pthread_attr_destroy()* to set *attr* to an implementation-defined invalid value. A
 31280 destroyed *attr* attributes object can be reinitialized using *pthread_attr_init()*; the results of
 31281 otherwise referencing the object after it has been destroyed are undefined.

31282 The *pthread_attr_init()* function shall initialize a thread attributes object *attr* with the default
 31283 value for all of the individual attributes used by a given implementation.

31284 The resulting attributes object (possibly modified by setting individual attribute values) when
 31285 used by *pthread_create()* defines the attributes of the thread created. A single attributes object can
 31286 be used in multiple simultaneous calls to *pthread_create()*. Results are undefined if
 31287 *pthread_attr_init()* is called specifying an already initialized *attr* attributes object.

31288 RETURN VALUE

31289 Upon successful completion, *pthread_attr_destroy()* and *pthread_attr_init()* shall return a value of
 31290 0; otherwise, an error number shall be returned to indicate the error.

31291 ERRORS

31292 The *pthread_attr_init()* function shall fail if:

31293 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

31294 These functions shall not return an error code of [EINTR].

31295 EXAMPLES

31296 None.

31297 APPLICATION USAGE

31298 None.

31299 RATIONALE

31300 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to
 31301 support probable future standardization in these areas without requiring that the function itself
 31302 be changed.

31303 Attributes objects provide clean isolation of the configurable aspects of threads. For example,
 31304 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When
 31305 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects
 31306 can help by allowing the changes to be isolated in a single place, rather than being spread across
 31307 every instance of thread creation.

31308 Attributes objects can be used to set up “classes” of threads with similar attributes; for example,
 31309 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes
 31310 can be defined in a single place and then referenced wherever threads need to be created.
 31311 Changes to “class” decisions become straightforward, and detailed analysis of each
 31312 *pthread_create()* call is not required.

31313 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had
 31314 been specified as structures, adding new attributes would force recompilation of all multi-

threaded programs when the attributes objects are extended; this might not be possible if different program components were supplied by different vendors.

Additionally, opaque attributes objects present opportunities for improving performance. Argument validity can be checked once when attributes are set, rather than each time a thread is created. Implementations often need to cache kernel objects that are expensive to create. Opaque attributes objects provide an efficient mechanism to detect when cached objects become invalid due to attribute changes.

Since assignment is not necessarily defined on a given opaque type, implementation-defined default values cannot be defined in a portable way. The solution to this problem is to allow attributes objects to be initialized dynamically by attributes object initialization functions, so that default values can be supplied automatically by the implementation.

The following proposal was provided as a suggested alternative to the supplied attributes:

1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to the initialization routines (*pthread_create()*, *pthread_mutex_init()*, *pthread_cond_init()*). The parameter containing the flags should be an opaque type for extensibility. If no flags are set in the parameter, then the objects are created with default characteristics. An implementation may specify implementation-defined flag values and associated behavior.
2. If further specialization of mutexes and condition variables is necessary, implementations may specify additional procedures that operate on the **pthread_mutex_t** and **pthread_cond_t** objects (instead of on attributes objects).

The difficulties with this solution are:

1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**, application programmers need to know the location of each bit. If bits are set or read by encapsulation (that is, *get* and *set* functions), then the bitmask is merely an implementation of attributes objects as currently defined and should not be exposed to the programmer.
2. Many attributes are not Boolean or very small integral values. For example, scheduling policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the bitmask can only reasonably control whether particular attributes are set or not, and it cannot serve as the repository of the value itself. The value needs to be specified as a function parameter (which is non-extensible), or by setting a structure field (which is non-opaque), or by *get* and *set* functions (making the bitmask a redundant addition to the attributes objects).

Stack size is defined as an optional attribute because the very notion of a stack is inherently machine-dependent. Some implementations may not be able to change the size of the stack, for example, and others may not need to because stack pages may be discontinuous and can be allocated and released on demand.

The attribute mechanism has been designed in large measure for extensibility. Future extensions to the attribute mechanism or to any attributes object defined in this volume of IEEE Std 1003.1-2001 has to be done with care so as not to affect binary-compatibility.

Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*, may have their size fixed at compile time. This means, for example, a *pthread_create()* in an implementation with extensions to **pthread_attr_t** cannot look beyond the area that the binary application assumes is valid. This suggests that implementations should maintain a size field in the attributes object, as well as possibly version information, if extensions in different directions

31362 (possibly by different vendors) are to be accommodated.

31363 **FUTURE DIRECTIONS**

31364 None.

31365 **SEE ALSO**

31366 *pthread_attr_getstackaddr()*, *pthread_attr_getstacksize()*, *pthread_attr_getdetachstate()*,
31367 *pthread_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

31368 **CHANGE HISTORY**

31369 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31370 **Issue 6**

31371 The *pthread_attr_destroy()* and *pthread_attr_init()* functions are marked as part of the Threads
31372 option.

31373 IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already
31374 initialized thread attributes object is undefined.

31375 **NAME**

31376 pthread_attr_getdetachstate, pthread_attr_setdetachstate — get and set the detachstate attribute

31377 **SYNOPSIS**

31378 THR #include <pthread.h>

```

31379 int pthread_attr_getdetachstate(const pthread_attr_t *attr,
31380 int *detachstate);
31381 int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
31382

```

31383 **DESCRIPTION**

31384 The *detachstate* attribute controls whether the thread is created in a detached state. If the thread
 31385 is created detached, then use of the ID of the newly created thread by the *pthread_detach()* or
 31386 *pthread_join()* function is an error.

31387 The *pthread_attr_getdetachstate()* and *pthread_attr_setdetachstate()* functions, respectively, shall
 31388 get and set the *detachstate* attribute in the *attr* object.

31389 For *pthread_attr_getdetachstate()*, *detachstate* shall be set to either
 31390 PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

31391 For *pthread_attr_setdetachstate()*, the application shall set *detachstate* to either
 31392 PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

31393 A value of PTHREAD_CREATE_DETACHED shall cause all threads created with *attr* to be in
 31394 the detached state, whereas using a value of PTHREAD_CREATE_JOINABLE shall cause all
 31395 threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute
 31396 shall be PTHREAD_CREATE_JOINABLE.

31397 **RETURN VALUE**

31398 Upon successful completion, *pthread_attr_getdetachstate()* and *pthread_attr_setdetachstate()* shall
 31399 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31400 The *pthread_attr_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate*
 31401 if successful.

31402 **ERRORS**

31403 The *pthread_attr_setdetachstate()* function shall fail if:

31404 [EINVAL] The value of *detachstate* was not valid

31405 These functions shall not return an error code of [EINTR].

31406 **EXAMPLES**

31407 None.

31408 **APPLICATION USAGE**

31409 None.

31410 **RATIONALE**

31411 None.

31412 **FUTURE DIRECTIONS**

31413 None.

31414 **SEE ALSO**

31415 *pthread_attr_destroy()*, *pthread_attr_getstackaddr()*, *pthread_attr_getstacksize()*, *pthread_create()*, the
 31416 Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

31417 **CHANGE HISTORY**

31418 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31419 **Issue 6**

31420 The *pthread_attr_setdetachstate()* and *pthread_attr_getdetachstate()* functions are marked as part of
31421 the Threads option.

31422 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

31423 **NAME**

31424 pthread_attr_getguardsize, pthread_attr_setguardsize — get and set the thread guardsize
 31425 attribute

31426 **SYNOPSIS**

31427 XSI `#include <pthread.h>`

```
31428 int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
31429                             size_t *restrict guardsize);
31430 int pthread_attr_setguardsize(pthread_attr_t *attr,
31431                             size_t guardsize);
31432
```

31433 **DESCRIPTION**

31434 The *pthread_attr_getguardsize()* function shall get the *guardsize* attribute in the *attr* object. This
 31435 attribute shall be returned in the *guardsize* parameter.

31436 The *pthread_attr_setguardsize()* function shall set the *guardsize* attribute in the *attr* object. The new
 31437 value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard
 31438 area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard
 31439 area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

31440 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The
 31441 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is
 31442 created with guard protection, the implementation allocates extra memory at the overflow end
 31443 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows
 31444 into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

31445 A conforming implementation may round up the value contained in *guardsize* to a multiple of
 31446 the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an implementation
 31447 rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread_attr_getguardsize()*
 31448 specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous
 31449 *pthread_attr_setguardsize()* function call.

31450 The default value of the *guardsize* attribute is {PAGESIZE} bytes. The actual value of {PAGESIZE}
 31451 is implementation-defined.

31452 If the *stackaddr* or *stack* attribute has been set (that is, the caller is allocating and managing its
 31453 own thread stacks), the *guardsize* attribute shall be ignored and no protection shall be provided
 31454 by the implementation. It is the responsibility of the application to manage stack overflow along
 31455 with stack allocation and management in this case.

31456 **RETURN VALUE**

31457 If successful, the *pthread_attr_getguardsize()* and *pthread_attr_setguardsize()* functions shall return
 31458 zero; otherwise, an error number shall be returned to indicate the error.

31459 **ERRORS**

31460 The *pthread_attr_getguardsize()* and *pthread_attr_setguardsize()* functions shall fail if:

31461 [EINVAL] The attribute *attr* is invalid.

31462 [EINVAL] The parameter *guardsize* is invalid.

31463 These functions shall not return an error code of [EINTR].

31464 **EXAMPLES**

31465 None.

31466 **APPLICATION USAGE**

31467 None.

31468 **RATIONALE**31469 The *guardsize* attribute is provided to the application for two reasons:

- 31470 1. Overflow protection can potentially result in wasted system resources. An application
31471 that creates a large number of threads, and which knows its threads never overflow their
31472 stack, can save system resources by turning off guard areas.
- 31473 2. When threads allocate large data structures on the stack, large guard areas may be needed
31474 to detect stack overflow.

31475 **FUTURE DIRECTIONS**

31476 None.

31477 **SEE ALSO**31478 The Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**, **<sys/mman.h>**31479 **CHANGE HISTORY**

31480 First released in Issue 5.

31481 **Issue 6**

31482 In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the
31483 second error condition.

31484 The **restrict** keyword is added to the *pthread_attr_getguardsize()* prototype for alignment with the
31485 ISO / IEC 9899:1999 standard.

31486 **NAME**

31487 pthread_attr_getinheritsched, pthread_attr_setinheritsched — get and set the inheritsched
 31488 attribute (**REALTIME THREADS**)

31489 **SYNOPSIS**

31490 THR TPS #include <pthread.h>

```
31491 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
31492 int *restrict inheritsched);
31493 int pthread_attr_setinheritsched(pthread_attr_t *attr,
31494 int inheritsched);
31495
```

31496 **DESCRIPTION**

31497 The *pthread_attr_getinheritsched()*, and *pthread_attr_setinheritsched()* functions, respectively, shall
 31498 get and set the *inheritsched* attribute in the *attr* argument.

31499 When the attributes objects are used by *pthread_create()*, the *inheritsched* attribute determines
 31500 how the other scheduling attributes of the created thread shall be set.

31501 **PTHREAD_INHERIT_SCHED**

31502 Specifies that the thread scheduling attributes shall be inherited from the creating thread,
 31503 and the scheduling attributes in this *attr* argument shall be ignored.

31504 **PTHREAD_EXPLICIT_SCHED**

31505 Specifies that the thread scheduling attributes shall be set to the corresponding values from
 31506 this attributes object.

31507 The symbols **PTHREAD_INHERIT_SCHED** and **PTHREAD_EXPLICIT_SCHED** are defined in
 31508 the **<pthread.h>** header.

31509 The following thread scheduling attributes defined by IEEE Std 1003.1-2001 are affected by the
 31510 *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and
 31511 scheduling contention scope (*contentionscope*).

31512 **RETURN VALUE**

31513 If successful, the *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions shall
 31514 return zero; otherwise, an error number shall be returned to indicate the error.

31515 **ERRORS**

31516 The *pthread_attr_setinheritsched()* function may fail if:

31517 [EINVAL] The value of *inheritsched* is not valid.

31518 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31519 These functions shall not return an error code of [EINTR].

31520 **EXAMPLES**

31521 None.

31522 **APPLICATION USAGE**

31523 After these attributes have been set, a thread can be created with the specified attributes using
 31524 *pthread_create()*. Using these routines does not affect the current running thread.

31525 **RATIONALE**

31526 None.

31527 **FUTURE DIRECTIONS**

31528 None.

31529 **SEE ALSO**

31530 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getschedpolicy()*,
31531 *pthread_attr_getschedparam()*, *pthread_create()*, the Base Definitions volume of
31532 IEEE Std 1003.1-2001, <**pthread.h**>, <**sched.h**>

31533 **CHANGE HISTORY**

31534 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31535 Marked as part of the Realtime Threads Feature Group.

31536 **Issue 6**

31537 The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions are marked as part
31538 of the Threads and Thread Execution Scheduling options.

31539 The [ENOSYS] error condition has been removed as stubs need not be provided if an
31540 implementation does not support the Thread Execution Scheduling option.

31541 The **restrict** keyword is added to the *pthread_attr_getinheritsched()* prototype for alignment with
31542 the ISO / IEC 9899:1999 standard.

31543 **NAME**

31544 pthread_attr_getschedparam, pthread_attr_setschedparam — get and set the schedparam
 31545 attribute

31546 **SYNOPSIS**

31547 THR #include <pthread.h>

```
31548 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
31549 struct sched_param *restrict param);
31550 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
31551 const struct sched_param *restrict param);
31552
```

31553 **DESCRIPTION**

31554 The *pthread_attr_getschedparam()*, and *pthread_attr_setschedparam()* functions, respectively, shall
 31555 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*
 31556 structure are defined in the <**sched.h**> header. For the SCHED_FIFO and SCHED_RR policies,
 31557 the only required member of *param* is *sched_priority*.

31558 TSP For the SCHED_SPORADIC policy, the required members of the *param* structure are
 31559 *sched_priority*, *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and
 31560 *sched_ss_max_repl*. The specified *sched_ss_repl_period* must be greater than or equal to the
 31561 specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.
 31562 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
 31563 function to succeed; if not, the function shall fail.

31564 **RETURN VALUE**

31565 If successful, the *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions shall
 31566 return zero; otherwise, an error number shall be returned to indicate the error.

31567 **ERRORS**

31568 The *pthread_attr_setschedparam()* function may fail if:

31569 [EINVAL] The value of *param* is not valid.

31570 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31571 These functions shall not return an error code of [EINTR].

31572 **EXAMPLES**

31573 None.

31574 **APPLICATION USAGE**

31575 After these attributes have been set, a thread can be created with the specified attributes using
 31576 *pthread_create()*. Using these routines does not affect the current running thread.

31577 **RATIONALE**

31578 None.

31579 **FUTURE DIRECTIONS**

31580 None.

31581 **SEE ALSO**

31582 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getinheritsched()*,
 31583 *pthread_attr_getschedpolicy()*, *pthread_create()*, the Base Definitions volume of
 31584 IEEE Std 1003.1-2001, <**pthread.h**>, <**sched.h**>

CHANGE HISTORY

31585
31586 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

31588 The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions are marked as part
31589 of the Threads option.

31590 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

31591 The **restrict** keyword is added to the *pthread_attr_getschedparam()* and
31592 *pthread_attr_setschedparam()* prototypes for alignment with the ISO / IEC 9899:1999 standard.

31593 **NAME**

31594 pthread_attr_getschedpolicy, pthread_attr_setschedpolicy — get and set the schedpolicy
 31595 attribute (**REALTIME THREADS**)

31596 **SYNOPSIS**

31597 THR TPS #include <pthread.h>

```
31598 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
31599 int *restrict policy);
31600 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
31601
```

31602 **DESCRIPTION**

31603 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions, respectively, shall
 31604 get and set the *schedpolicy* attribute in the *attr* argument.

31605 The supported values of *policy* shall include SCHED_FIFO, SCHED_RR, and SCHED_OTHER,
 31606 which are defined in the <sched.h> header. When threads executing with the scheduling policy
 31607 TSP SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on a mutex, they shall acquire
 31608 the mutex in priority order when the mutex is unlocked.

31609 **RETURN VALUE**

31610 If successful, the *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions shall
 31611 return zero; otherwise, an error number shall be returned to indicate the error.

31612 **ERRORS**

31613 The *pthread_attr_setschedpolicy()* function may fail if:

31614 [EINVAL] The value of *policy* is not valid.

31615 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31616 These functions shall not return an error code of [EINTR].

31617 **EXAMPLES**

31618 None.

31619 **APPLICATION USAGE**

31620 After these attributes have been set, a thread can be created with the specified attributes using
 31621 *pthread_create()*. Using these routines does not affect the current running thread.

31622 **RATIONALE**

31623 None.

31624 **FUTURE DIRECTIONS**

31625 None.

31626 **SEE ALSO**

31627 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getinheritsched()*,
 31628 *pthread_attr_getschedparam()*, *pthread_create()*, the Base Definitions volume of
 31629 IEEE Std 1003.1-2001, <pthread.h>, <sched.h>

31630 **CHANGE HISTORY**

31631 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31632 Marked as part of the Realtime Threads Feature Group.

31633 **Issue 6**

31634 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions are marked as part of
31635 the Threads and Thread Execution Scheduling options.

31636 The [ENOSYS] error condition has been removed as stubs need not be provided if an
31637 implementation does not support the Thread Execution Scheduling option.

31638 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

31639 The **restrict** keyword is added to the *pthread_attr_getschedpolicy()* prototype for alignment with
31640 the ISO / IEC 9899:1999 standard.

31641 **NAME**

31642 pthread_attr_getscope, pthread_attr_setscope — get and set the contentionscope attribute
 31643 (**REALTIME THREADS**)

31644 **SYNOPSIS**

31645 THR TPS #include <pthread.h>

```
31646 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
31647 int *restrict contentionscope);
31648 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
31649
```

31650 **DESCRIPTION**

31651 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions, respectively, shall get and set
 31652 the *contentionscope* attribute in the *attr* object.

31653 The *contentionscope* attribute may have the values PTHREAD_SCOPE_SYSTEM, signifying
 31654 system scheduling contention scope, or PTHREAD_SCOPE_PROCESS, signifying process
 31655 scheduling contention scope. The symbols PTHREAD_SCOPE_SYSTEM and
 31656 PTHREAD_SCOPE_PROCESS are defined in the <pthread.h> header.

31657 **RETURN VALUE**

31658 If successful, the *pthread_attr_getscope()* and *pthread_attr_setscope()* functions shall return zero;
 31659 otherwise, an error number shall be returned to indicate the error.

31660 **ERRORS**

31661 The *pthread_attr_setscope()* function may fail if:

31662 [EINVAL] The value of *contentionscope* is not valid.

31663 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31664 These functions shall not return an error code of [EINTR].

31665 **EXAMPLES**

31666 None.

31667 **APPLICATION USAGE**

31668 After these attributes have been set, a thread can be created with the specified attributes using
 31669 *pthread_create()*. Using these routines does not affect the current running thread.

31670 **RATIONALE**

31671 None.

31672 **FUTURE DIRECTIONS**

31673 None.

31674 **SEE ALSO**

31675 *pthread_attr_destroy()*, *pthread_attr_getinheritsched()*, *pthread_attr_getschedpolicy()*,
 31676 *pthread_attr_getschedparam()*, *pthread_create()*, the Base Definitions volume of
 31677 IEEE Std 1003.1-2001, <pthread.h>, <sched.h>

31678 **CHANGE HISTORY**

31679 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31680 Marked as part of the Realtime Threads Feature Group.

31681 **Issue 6**

31682 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions are marked as part of the
31683 Threads and Thread Execution Scheduling options.

31684 The [ENOSYS] error condition has been removed as stubs need not be provided if an
31685 implementation does not support the Thread Execution Scheduling option.

31686 The **restrict** keyword is added to the *pthread_attr_getscope()* prototype for alignment with the
31687 ISO / IEC 9899:1999 standard.

31688 **NAME**

31689 pthread_attr_getstack, pthread_attr_setstack — get and set stack attributes

31690 **SYNOPSIS**

31691 THR #include <pthread.h>

```

31692 TSA TSS int pthread_attr_getstack(const pthread_attr_t *restrict attr,
31693 void **restrict stackaddr, size_t *restrict stacksize);
31694 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
31695 size_t stacksize);
31696

```

31697 **DESCRIPTION**

31698 The *pthread_attr_getstack()* and *pthread_attr_setstack()* functions, respectively, shall get and set
 31699 the thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

31700 The stack attributes specify the area of storage to be used for the created thread's stack. The base
 31701 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be
 31702 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD_STACK_MIN}. The *stackaddr* shall be
 31703 aligned appropriately to be used as a stack; for example, *pthread_attr_setstack()* may fail with
 31704 [EINVAL] if (*stackaddr* & 0x7) is not 0. All pages within the stack described by *stackaddr* and
 31705 *stacksize* shall be both readable and writable by the thread.

31706 **RETURN VALUE**

31707 Upon successful completion, these functions shall return a value of 0; otherwise, an error
 31708 number shall be returned to indicate the error.

31709 The *pthread_attr_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*
 31710 if successful.

31711 **ERRORS**

31712 The *pthread_attr_setstack()* function shall fail if:

31713 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds an
 31714 implementation-defined limit.

31715 The *pthread_attr_setstack()* function may fail if:

31716 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or
 31717 if (*stackaddr* + *stacksize*) lacks proper alignment.

31718 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable
 31719 and writable by the thread.

31720 These functions shall not return an error code of [EINTR].

31721 **EXAMPLES**

31722 None.

31723 **APPLICATION USAGE**

31724 These functions are appropriate for use by applications in an environment where the stack for a
 31725 thread must be placed in some particular region of memory.

31726 While it might seem that an application could detect stack overflow by providing a protected
 31727 page outside the specified stack region, this cannot be done portably. Implementations are free
 31728 to place the thread's initial stack pointer anywhere within the specified region to accommodate
 31729 the machine's stack pointer behavior and allocation requirements. Furthermore, on some
 31730 architectures, such as the IA-64, "overflow" might mean that two separate stack pointers
 31731 allocated within the region will overlap somewhere in the middle of the region.

31732 **RATIONALE**

31733 None.

31734 **FUTURE DIRECTIONS**

31735 None.

31736 **SEE ALSO**

31737 *pthread_attr_init()*, *pthread_attr_setdetachstate()*, *pthread_attr_setstacksize()*, *pthread_create()*, the
31738 Base Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <pthread.h>

31739 **CHANGE HISTORY**

31740 First released in Issue 6. Developed as an XSI extension and brought into the BASE by IEEE
31741 PASC Interpretation 1003.1 #101.

31742 **NAME**

31743 pthread_attr_getstackaddr, pthread_attr_setstackaddr — get and set the stackaddr attribute

31744 **SYNOPSIS**

31745 THR TSA #include <pthread.h>

```

31746 OB      int pthread_attr_getstackaddr(const pthread_attr_t *restrict attr,
31747      void **restrict stackaddr);
31748      int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
31749

```

31750 **DESCRIPTION**

31751 The *pthread_attr_getstackaddr()* and *pthread_attr_setstackaddr()* functions, respectively, shall get
 31752 and set the thread creation *stackaddr* attribute in the *attr* object.

31753 The *stackaddr* attribute specifies the location of storage to be used for the created thread's stack.
 31754 The size of the storage shall be at least {PTHREAD_STACK_MIN}.

31755 **RETURN VALUE**

31756 Upon successful completion, *pthread_attr_getstackaddr()* and *pthread_attr_setstackaddr()* shall
 31757 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31758 The *pthread_attr_getstackaddr()* function stores the *stackaddr* attribute value in *stackaddr* if
 31759 successful.

31760 **ERRORS**

31761 No errors are defined.

31762 These functions shall not return an error code of [EINTR].

31763 **EXAMPLES**

31764 None.

31765 **APPLICATION USAGE**

31766 The specification of the *stackaddr* attribute presents several ambiguities that make portable use of
 31767 these interfaces impossible. The description of the single address parameter as a “stack” does
 31768 not specify a particular relationship between the address and the “stack” implied by that
 31769 address. For example, the address may be taken as the low memory address of a buffer intended
 31770 for use as a stack, or it may be taken as the address to be used as the initial stack pointer register
 31771 value for the new thread. These two are not the same except for a machine on which the stack
 31772 grows “up” from low memory to high, and on which a “push” operation first stores the value in
 31773 memory and then increments the stack pointer register. Further, on a machine where the stack
 31774 grows “down” from high memory to low, interpretation of the address as the “low memory”
 31775 address requires a determination of the intended size of the stack. IEEE Std 1003.1-2001 has
 31776 introduced the new interfaces *pthread_attr_setstack()* and *pthread_attr_getstack()* to resolve these
 31777 ambiguities.

31778 **RATIONALE**

31779 None.

31780 **FUTURE DIRECTIONS**

31781 None.

31782 **SEE ALSO**

31783 *pthread_attr_destroy()*, *pthread_attr_getdetachstate()*, *pthread_attr_getstack()*,
 31784 *pthread_attr_getstacksize()*, *pthread_attr_setstack()*, *pthread_create()*, the Base Definitions volume
 31785 of IEEE Std 1003.1-2001, <limits.h>, <pthread.h>

31786 **CHANGE HISTORY**

31787 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31788 **Issue 6**

31789 The *pthread_attr_getstackaddr()* and *pthread_attr_setstackaddr()* functions are marked as part of
31790 the Threads and Thread Stack Address Attribute options.

31791 The **restrict** keyword is added to the *pthread_attr_getstackaddr()* prototype for alignment with the
31792 ISO / IEC 9899:1999 standard.

31793 These functions are marked obsolescent.

31794 **NAME**

31795 pthread_attr_getstacksize, pthread_attr_setstacksize — get and set the stacksize attribute

31796 **SYNOPSIS**

31797 THR TSS #include <pthread.h>

```
31798 int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
31799                             size_t *restrict stacksize);
31800 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
31801
```

31802 **DESCRIPTION**

31803 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions, respectively, shall get
31804 and set the thread creation *stacksize* attribute in the *attr* object.

31805 The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created
31806 threads stack.

31807 **RETURN VALUE**

31808 Upon successful completion, *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* shall
31809 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31810 The *pthread_attr_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if
31811 successful.

31812 **ERRORS**

31813 The *pthread_attr_setstacksize()* function shall fail if:

31814 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds a
31815 system-imposed limit.

31816 These functions shall not return an error code of [EINTR].

31817 **EXAMPLES**

31818 None.

31819 **APPLICATION USAGE**

31820 None.

31821 **RATIONALE**

31822 None.

31823 **FUTURE DIRECTIONS**

31824 None.

31825 **SEE ALSO**

31826 *pthread_attr_destroy()*, *pthread_attr_getstackaddr()*, *pthread_attr_getdetachstate()*, *pthread_create()*,
31827 the Base Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <pthread.h>

31828 **CHANGE HISTORY**

31829 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31830 **Issue 6**

31831 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions are marked as part of the
31832 Threads and Thread Stack Size Attribute options.

31833 The **restrict** keyword is added to the *pthread_attr_getstacksize()* prototype for alignment with the
31834 ISO / IEC 9899:1999 standard.

31835	IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 43 is applied, correcting the margin code	
31836	in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from “Thread Stack	
31837	Address Attribute” option to “Thread Stack Size Attribute” option.	

31838 NAME

31839 pthread_attr_init — initialize the thread attributes object

31840 SYNOPSIS

31841 THR `#include <pthread.h>`

31842 `int pthread_attr_init(pthread_attr_t *attr);`

31843

31844 DESCRIPTION

31845 Refer to *pthread_attr_destroy()*.

31846 **NAME**

31847 pthread_attr_setdetachstate — set the detachstate attribute

31848 **SYNOPSIS**31849 THR `#include <pthread.h>`31850 `int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);`

31851

31852 **DESCRIPTION**31853 Refer to *pthread_attr_getdetachstate()*.

31854 **NAME**

31855 pthread_attr_setguardsize — set the thread guardsize attribute

31856 **SYNOPSIS**

31857 xSI #include <pthread.h>

31858 int pthread_attr_setguardsize(pthread_attr_t *attr,
31859 size_t guardsize);

31860

31861 **DESCRIPTION**

31862 Refer to *pthread_attr_getguardsize()*.

31863 **NAME**31864 pthread_attr_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)31865 **SYNOPSIS**

31866 THR TPS #include <pthread.h>

31867 int pthread_attr_setinheritsched(pthread_attr_t *attr,
31868 int inheritsched);

31869

31870 **DESCRIPTION**31871 Refer to *pthread_attr_getinheritsched()*.

31872 NAME

31873 pthread_attr_setschedparam — set the schedparam attribute

31874 SYNOPSIS

31875 THR #include <pthread.h>

```
31876 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,  
31877                               const struct sched_param *restrict param);
```

31878

31879 DESCRIPTION

31880 Refer to *pthread_attr_getschedparam()*.

31881 **NAME**

31882 pthread_attr_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)

31883 **SYNOPSIS**

31884 THR TPS `#include <pthread.h>`

31885 `int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);`

31886

31887 **DESCRIPTION**

31888 Refer to *pthread_attr_getschedpolicy()*.

31889 **NAME**

31890 pthread_attr_setscope — set the contentionscope attribute (**REALTIME THREADS**)

31891 **SYNOPSIS**

31892 THR TPS `#include <pthread.h>`

31893 `int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);`

31894

31895 **DESCRIPTION**

31896 Refer to *pthread_attr_getscope()*.

31897 **NAME**

31898 pthread_attr_setstack — set the stack attribute

31899 **SYNOPSIS**31900 xSI `#include <pthread.h>`31901 `int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,`
31902 `size_t stacksize);`

31903

31904 **DESCRIPTION**31905 Refer to *pthread_attr_getstack()*.

31906 **NAME**

31907 pthread_attr_setstackaddr — set the stackaddr attribute

31908 **SYNOPSIS**

31909 THR TSA #include <pthread.h>

31910 OB int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);

31911

31912 **DESCRIPTION**

31913 Refer to *pthread_attr_getstackaddr()*.

31914 **NAME**

31915 pthread_attr_setstacksize — set the stacksize attribute

31916 **SYNOPSIS**

31917 THR TSS #include <pthread.h>

31918 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);

31919

31920 **DESCRIPTION**31921 Refer to *pthread_attr_getstacksize()*.

31922 **NAME**

31923 pthread_barrier_destroy, pthread_barrier_init — destroy and initialize a barrier object
 31924 (**ADVANCED REALTIME THREADS**)

31925 **SYNOPSIS**

```
31926 THR BAR #include <pthread.h>

31927 int pthread_barrier_destroy(pthread_barrier_t *barrier);
31928 int pthread_barrier_init(pthread_barrier_t *restrict barrier,
31929     const pthread_barrierattr_t *restrict attr, unsigned count);
31930
```

31931 **DESCRIPTION**

31932 The *pthread_barrier_destroy()* function shall destroy the barrier referenced by *barrier* and release
 31933 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until
 31934 the barrier is reinitialized by another call to *pthread_barrier_init()*. An implementation may use
 31935 this function to set *barrier* to an invalid value. The results are undefined if
 31936 *pthread_barrier_destroy()* is called when any thread is blocked on the barrier, or if this function is
 31937 called with an uninitialized barrier.

31938 The *pthread_barrier_init()* function shall allocate any resources required to use the barrier
 31939 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is
 31940 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of
 31941 a default barrier attributes object. The results are undefined if *pthread_barrier_init()* is called
 31942 when any thread is blocked on the barrier (that is, has not returned from the
 31943 *pthread_barrier_wait()* call). The results are undefined if a barrier is used without first being
 31944 initialized. The results are undefined if *pthread_barrier_init()* is called specifying an already
 31945 initialized barrier.

31946 The *count* argument specifies the number of threads that must call *pthread_barrier_wait()* before
 31947 any of them successfully return from the call. The value specified by *count* must be greater than
 31948 zero.

31949 If the *pthread_barrier_init()* function fails, the barrier shall not be initialized and the contents of
 31950 *barrier* are undefined.

31951 Only the object referenced by *barrier* may be used for performing synchronization. The result of
 31952 referring to copies of that object in calls to *pthread_barrier_destroy()* or *pthread_barrier_wait()* is
 31953 undefined.

31954 **RETURN VALUE**

31955 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 31956 be returned to indicate the error.

31957 **ERRORS**

31958 The *pthread_barrier_destroy()* function may fail if:

31959 [EBUSY] The implementation has detected an attempt to destroy a barrier while it is in
 31960 use (for example, while being used in a *pthread_barrier_wait()* call) by another
 31961 thread.

31962 [EINVAL] The value specified by *barrier* is invalid.

31963 The *pthread_barrier_init()* function shall fail if:

31964 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

31965 [EINVAL] The value specified by *count* is equal to zero.

- 31966 [ENOMEM] Insufficient memory exists to initialize the barrier.
- 31967 The *pthread_barrier_init()* function may fail if:
- 31968 [EBUSY] The implementation has detected an attempt to reinitialize a barrier while it is
31969 in use (for example, while being used in a *pthread_barrier_wait()* call) by
31970 another thread.
- 31971 [EINVAL] The value specified by *attr* is invalid.
- 31972 These functions shall not return an error code of [EINTR].
- 31973 **EXAMPLES**
- 31974 None.
- 31975 **APPLICATION USAGE**
- 31976 The *pthread_barrier_destroy()* and *pthread_barrier_init()* functions are part of the Barriers option
31977 and need not be provided on all implementations.
- 31978 **RATIONALE**
- 31979 None.
- 31980 **FUTURE DIRECTIONS**
- 31981 None.
- 31982 **SEE ALSO**
- 31983 *pthread_barrier_wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>
- 31984 **CHANGE HISTORY**
- 31985 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

31986 **NAME**31987 pthread_barrier_wait — synchronize at a barrier (**ADVANCED REALTIME THREADS**)31988 **SYNOPSIS**

31989 THR BAR #include <pthread.h>

31990 int pthread_barrier_wait(pthread_barrier_t *barrier);

31991

31992 **DESCRIPTION**

31993 The *pthread_barrier_wait()* function shall synchronize participating threads at the barrier
 31994 referenced by *barrier*. The calling thread shall block until the required number of threads have
 31995 called *pthread_barrier_wait()* specifying the barrier.

31996 When the required number of threads have called *pthread_barrier_wait()* specifying the barrier,
 31997 the constant PTHREAD_BARRIER_SERIAL_THREAD shall be returned to one unspecified
 31998 thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall
 31999 be reset to the state it had as a result of the most recent *pthread_barrier_init()* function that
 32000 referenced it.

32001 The constant PTHREAD_BARRIER_SERIAL_THREAD is defined in <pthread.h> and its value
 32002 shall be distinct from any other value returned by *pthread_barrier_wait()*.

32003 The results are undefined if this function is called with an uninitialized barrier.

32004 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the
 32005 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the
 32006 required number of threads have not arrived at the barrier during the execution of the signal
 32007 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until
 32008 the thread in the signal handler returns from it, it is unspecified whether other threads may
 32009 proceed past the barrier once they have all reached it.

32010 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to
 32011 use the same processing resources from eventually making forward progress in its execution.
 32012 Eligibility for processing resources shall be determined by the scheduling policy.

32013 **RETURN VALUE**

32014 Upon successful completion, the *pthread_barrier_wait()* function shall return
 32015 PTHREAD_BARRIER_SERIAL_THREAD for a single (arbitrary) thread synchronized at the
 32016 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to
 32017 indicate the error.

32018 **ERRORS**

32019 The *pthread_barrier_wait()* function may fail if:

32020 [EINVAL] The value specified by *barrier* does not refer to an initialized barrier object.

32021 This function shall not return an error code of [EINTR].

32022 **EXAMPLES**

32023 None.

32024 **APPLICATION USAGE**

32025 Applications using this function may be subject to priority inversion, as discussed in the Base
 32026 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

32027 The *pthread_barrier_wait()* function is part of the Barriers option and need not be provided on all
 32028 implementations.

32029 **RATIONALE**

32030 None.

32031 **FUTURE DIRECTIONS**

32032 None.

32033 **SEE ALSO**32034 *pthread_barrier_destroy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>32035 **CHANGE HISTORY**

32036 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32037 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

32038 **NAME**

32039 pthread_barrierattr_destroy, pthread_barrierattr_init — destroy and initialize the barrier
32040 attributes object (**ADVANCED REALTIME THREADS**)

32041 **SYNOPSIS**

32042 THR BAR #include <pthread.h>

32043 int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);

32044 int pthread_barrierattr_init(pthread_barrierattr_t *attr);

32045

32046 **DESCRIPTION**

32047 The *pthread_barrierattr_destroy()* function shall destroy a barrier attributes object. A destroyed
32048 *attr* attributes object can be reinitialized using *pthread_barrierattr_init()*; the results of otherwise
32049 referencing the object after it has been destroyed are undefined. An implementation may cause
32050 *pthread_barrierattr_destroy()* to set the object referenced by *attr* to an invalid value.

32051 The *pthread_barrierattr_init()* function shall initialize a barrier attributes object *attr* with the
32052 default value for all of the attributes defined by the implementation.

32053 Results are undefined if *pthread_barrierattr_init()* is called specifying an already initialized *attr*
32054 attributes object.

32055 After a barrier attributes object has been used to initialize one or more barriers, any function
32056 affecting the attributes object (including destruction) shall not affect any previously initialized
32057 barrier.

32058 **RETURN VALUE**

32059 If successful, the *pthread_barrierattr_destroy()* and *pthread_barrierattr_init()* functions shall return
32060 zero; otherwise, an error number shall be returned to indicate the error.

32061 **ERRORS**

32062 The *pthread_barrierattr_destroy()* function may fail if:

32063 [EINVAL] The value specified by *attr* is invalid.

32064 The *pthread_barrierattr_init()* function shall fail if:

32065 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

32066 These functions shall not return an error code of [EINTR].

32067 **EXAMPLES**

32068 None.

32069 **APPLICATION USAGE**

32070 The *pthread_barrierattr_destroy()* and *pthread_barrierattr_init()* functions are part of the Barriers
32071 option and need not be provided on all implementations.

32072 **RATIONALE**

32073 None.

32074 **FUTURE DIRECTIONS**

32075 None.

32076 **SEE ALSO**

32077 *pthread_barrierattr_getpshared()*, *pthread_barrierattr_setpshared()*, the Base Definitions volume of
32078 IEEE Std 1003.1-2001, <pthread.h>.

32079 **CHANGE HISTORY**

32080 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32081 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

32082 **NAME**

32083 pthread_barrierattr_getpshared, pthread_barrierattr_setpshared — get and set the process-
 32084 shared attribute of the barrier attributes object (**ADVANCED REALTIME THREADS**)

32085 **SYNOPSIS**

```
32086 THR      #include <pthread.h>

32087 BAR TSH   int pthread_barrierattr_getpshared(const pthread_barrierattr_t *
32088             restrict attr, int *restrict pshared);
32089             int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
32090             int pshared);
32091
```

32092 **DESCRIPTION**

32093 The *pthread_barrierattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 32094 from the attributes object referenced by *attr*. The *pthread_barrierattr_setpshared()* function shall
 32095 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

32096 The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a barrier to be
 32097 operated upon by any thread that has access to the memory where the barrier is allocated. If the
 32098 *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the barrier shall only be operated
 32099 upon by threads created within the same process as the thread that initialized the barrier; if
 32100 threads of different processes attempt to operate on such a barrier, the behavior is undefined.
 32101 The default value of the attribute shall be PTHREAD_PROCESS_PRIVATE. Both constants
 32102 PTHREAD_PROCESS_SHARED and PTHREAD_PROCESS_PRIVATE are defined in
 32103 **<pthread.h>**.

32104 Additional attributes, their default values, and the names of the associated functions to get and
 32105 set those attribute values are implementation-defined.

32106 **RETURN VALUE**

32107 If successful, the *pthread_barrierattr_getpshared()* function shall return zero and store the value of
 32108 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,
 32109 an error number shall be returned to indicate the error.

32110 If successful, the *pthread_barrierattr_setpshared()* function shall return zero; otherwise, an error
 32111 number shall be returned to indicate the error.

32112 **ERRORS**

32113 These functions may fail if:

32114 [EINVAL] The value specified by *attr* is invalid.

32115 The *pthread_barrierattr_setpshared()* function may fail if:

32116 [EINVAL] The new value specified for the *process-shared* attribute is not one of the legal
 32117 values PTHREAD_PROCESS_SHARED or PTHREAD_PROCESS_PRIVATE.

32118 These functions shall not return an error code of [EINTR].

32119 EXAMPLES

32120 None.

32121 APPLICATION USAGE

32122 The *pthread_barrierattr_getpshared()* and *pthread_barrierattr_setpshared()* functions are part of the
32123 Barriers option and need not be provided on all implementations.

32124 RATIONALE

32125 None.

32126 FUTURE DIRECTIONS

32127 None.

32128 SEE ALSO

32129 *pthread_barrier_destroy()*, *pthread_barrierattr_destroy()*, *pthread_barrierattr_init()*, the Base
32130 Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

32131 CHANGE HISTORY

32132 First released in Issue 6. Derived from IEEE Std 1003.1j-2000

32133 NAME

32134 pthread_barrierattr_init — initialize the barrier attributes object (**ADVANCED REALTIME**
32135 **THREADS**)

32136 SYNOPSIS

32137 THR BAR #include <pthread.h>

32138 int pthread_barrierattr_init(pthread_barrierattr_t *attr);

32139

32140 DESCRIPTION

32141 Refer to *pthread_barrierattr_destroy()*.

32142 **NAME**

32143 pthread_barrierattr_setpshared — set the process-shared attribute of the barrier attributes object
32144 (**ADVANCED REALTIME THREADS**)

32145 **SYNOPSIS**

32146 THR #include <pthread.h>

32147 BAR TSH int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
32148 int pshared);

32149

32150 **DESCRIPTION**

32151 Refer to *pthread_barrierattr_getpshared()*.

32152 **NAME**

32153 pthread_cancel — cancel execution of a thread

32154 **SYNOPSIS**

32155 THR #include <pthread.h>

32156 int pthread_cancel(pthread_t thread);

32157

32158 **DESCRIPTION**

32159 The *pthread_cancel()* function shall request that *thread* be canceled. The target thread's
 32160 cancelability state and type determines when the cancellation takes effect. When the cancellation
 32161 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last
 32162 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called
 32163 for *thread*. When the last destructor function returns, *thread* shall be terminated.

32164 The cancellation processing in the target thread shall run asynchronously with respect to the
 32165 calling thread returning from *pthread_cancel()*.

32166 **RETURN VALUE**

32167 If successful, the *pthread_cancel()* function shall return zero; otherwise, an error number shall be
 32168 returned to indicate the error.

32169 **ERRORS**32170 The *pthread_cancel()* function may fail if:

32171 [ESRCH] No thread could be found corresponding to that specified by the given thread
 32172 ID.

32173 The *pthread_cancel()* function shall not return an error code of [EINTR].32174 **EXAMPLES**

32175 None.

32176 **APPLICATION USAGE**

32177 None.

32178 **RATIONALE**

32179 Two alternative functions were considered for sending the cancellation notification to a thread.
 32180 One would be to define a new SIGCANCEL signal that had the cancellation semantics when
 32181 delivered; the other was to define the new *pthread_cancel()* function, which would trigger the
 32182 cancellation semantics.

32183 The advantage of a new signal was that so much of the delivery criteria were identical to that
 32184 used when trying to deliver a signal that making cancellation notification a signal was seen as
 32185 consistent. Indeed, many implementations implement cancellation using a special signal. On the
 32186 other hand, there would be no signal functions that could be used with this signal except
 32187 *pthread_kill()*, and the behavior of the delivered cancellation signal would be unlike any
 32188 previously existing defined signal.

32189 The benefits of a special function include the recognition that this signal would be defined
 32190 because of the similar delivery criteria and that this is the only common behavior between a
 32191 cancellation request and a signal. In addition, the cancellation delivery mechanism does not
 32192 have to be implemented as a signal. There are also strong, if not stronger, parallels with
 32193 language exception mechanisms than with signals that are potentially obscured if the delivery
 32194 mechanism is visibly closer to signals.

32195 In the end, it was considered that as there were so many exceptions to the use of the new signal
 32196 with existing signals functions it would be misleading. A special function has resolved this

32197 problem. This function was carefully defined so that an implementation wishing to provide the
32198 cancellation functions on top of signals could do so. The special function also means that
32199 implementations are not obliged to implement cancellation with signals.

32200 **FUTURE DIRECTIONS**

32201 None.

32202 **SEE ALSO**

32203 *pthread_exit()*, *pthread_cond_timedwait()*, *pthread_join()*, *pthread_setcancelstate()*, the Base
32204 Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

32205 **CHANGE HISTORY**

32206 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32207 **Issue 6**

32208 The *pthread_cancel()* function is marked as part of the Threads option.

32209 **NAME**

32210 pthread_cleanup_pop, pthread_cleanup_push — establish cancellation handlers |

32211 **SYNOPSIS**

32212 THR #include <pthread.h>

32213 void pthread_cleanup_pop(int execute);

32214 void pthread_cleanup_push(void (*routine)(void*), void *arg);

32215

32216 **DESCRIPTION**32217 The *pthread_cleanup_pop()* function shall remove the routine at the top of the calling thread's
32218 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero). |32219 The *pthread_cleanup_push()* function shall push the specified cancellation cleanup handler *routine*
32220 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be
32221 popped from the cancellation cleanup stack and invoked with the argument *arg* when: |32222 • The thread exits (that is, calls *pthread_exit()*).

32223 • The thread acts upon a cancellation request. |

32224 • The thread calls *pthread_cleanup_pop()* with a non-zero *execute* argument.32225 These functions may be implemented as macros. The application shall ensure that they appear
32226 as statements, and in pairs within the same lexical scope (that is, the *pthread_cleanup_push()*
32227 macro may be thought to expand to a token list whose first token is '{' with
32228 *pthread_cleanup_pop()* expanding to a token list whose last token is the corresponding '}').32229 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to
32230 *pthread_cleanup_push()* or *pthread_cleanup_pop()* made without the matching call since the jump
32231 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancellation
32232 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation
32233 cleanup handler. |32234 **RETURN VALUE**32235 The *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions shall not return a value.32236 **ERRORS**

32237 No errors are defined.

32238 These functions shall not return an error code of [EINTR].

32239 **EXAMPLES**32240 The following is an example using thread primitives to implement a cancelable, writers-priority
32241 read-write lock:

```

32242 typedef struct {
32243     pthread_mutex_t lock;
32244     pthread_cond_t rcond,
32245     wcond;
32246     int lock_count; /* < 0 .. Held by writer. */
32247                   /* > 0 .. Held by lock_count readers. */
32248                   /* = 0 .. Held by nobody. */
32249     int waiting_writers; /* Count of waiting writers. */
32250 } rwlock;

32251 void
32252 waiting_reader_cleanup(void *arg)
32253 {

```

```

32254         rwlock *l;
32255         l = (rwlock *) arg;
32256         pthread_mutex_unlock(&l->lock);
32257     }
32258
32258 void
32259 lock_for_read(rwlock *l)
32260 {
32261     pthread_mutex_lock(&l->lock);
32262     pthread_cleanup_push(waiting_reader_cleanup, l);
32263     while ((l->lock_count < 0) && (l->waiting_writers != 0))
32264         pthread_cond_wait(&l->rcond, &l->lock);
32265     l->lock_count++;
32266     /*
32267      * Note the pthread_cleanup_pop executes
32268      * waiting_reader_cleanup.
32269      */
32270     pthread_cleanup_pop(1);
32271 }
32272
32272 void
32273 release_read_lock(rwlock *l)
32274 {
32275     pthread_mutex_lock(&l->lock);
32276     if (--l->lock_count == 0)
32277         pthread_cond_signal(&l->wcond);
32278     pthread_mutex_unlock(l);
32279 }
32280
32280 void
32281 waiting_writer_cleanup(void *arg)
32282 {
32283     rwlock *l;
32284
32285     l = (rwlock *) arg;
32286     if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
32287         /*
32288          * This only happens if we have been canceled.
32289          */
32290         pthread_cond_broadcast(&l->wcond);
32291     }
32292     pthread_mutex_unlock(&l->lock);
32293 }
32294
32294 void
32295 lock_for_write(rwlock *l)
32296 {
32297     pthread_mutex_lock(&l->lock);
32298     l->waiting_writers++;
32299     pthread_cleanup_push(waiting_writer_cleanup, l);
32300     while (l->lock_count != 0)
32301         pthread_cond_wait(&l->wcond, &l->lock);
32302     l->lock_count = -1;
32303     /*

```

```

32303         * Note the pthread_cleanup_pop executes
32304         * waiting_writer_cleanup.
32305         */
32306         pthread_cleanup_pop(1);
32307     }

32308 void
32309 release_write_lock(rwlock *l)
32310 {
32311     pthread_mutex_lock(&l->lock);
32312     l->lock_count = 0;
32313     if (l->waiting_writers == 0)
32314         pthread_cond_broadcast(&l->rcond)
32315     else
32316         pthread_cond_signal(&l->wcond);
32317     pthread_mutex_unlock(&l->lock);
32318 }

32319 /*
32320  * This function is called to initialize the read/write lock.
32321  */
32322 void
32323 initialize_rwlock(rwlock *l)
32324 {
32325     pthread_mutex_init(&l->lock, pthread_mutexattr_default);
32326     pthread_cond_init(&l->wcond, pthread_condattr_default);
32327     pthread_cond_init(&l->rcond, pthread_condattr_default);
32328     l->lock_count = 0;
32329     l->waiting_writers = 0;
32330 }

32331 reader_thread()
32332 {
32333     lock_for_read(&lock);
32334     pthread_cleanup_push(release_read_lock, &lock);
32335     /*
32336      * Thread has read lock.
32337      */
32338     pthread_cleanup_pop(1);
32339 }

32340 writer_thread()
32341 {
32342     lock_for_write(&lock);
32343     pthread_cleanup_push(release_write_lock, &lock);
32344     /*
32345      * Thread has write lock.
32346      */
32347     pthread_cleanup_pop(1);
32348 }

```


32349 **APPLICATION USAGE**

32350 The two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push()* and
 32351 *pthread_cleanup_pop()*, can be thought of as left and right parentheses. They always need to be
 32352 matched.

32353 **RATIONALE**

32354 The restriction that the two routines that push and pop cancellation cleanup handlers,
 32355 *pthread_cleanup_push()* and *pthread_cleanup_pop()*, have to appear in the same lexical scope
 32356 allows for efficient macro or compiler implementations and efficient storage management. A
 32357 sample implementation of these routines as macros might look like this:

```
32358 #define pthread_cleanup_push(rtn,arg) { \
32359     struct _pthread_handler_rec __cleanup_handler, *__head; \
32360     __cleanup_handler.rtn = rtn; \
32361     __cleanup_handler.arg = arg; \
32362     (void) pthread_getspecific(_pthread_handler_key, &__head); \
32363     __cleanup_handler.next = *__head; \
32364     *__head = &__cleanup_handler;

32365 #define pthread_cleanup_pop(ex) \
32366     *__head = __cleanup_handler.next; \
32367     if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
32368 }
```

32369 A more ambitious implementation of these routines might do even better by allowing the
 32370 compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

32371 This volume of IEEE Std 1003.1-2001 currently leaves unspecified the effect of calling *longjmp()*
 32372 from a signal handler executing in a POSIX System Interfaces function. If an implementation
 32373 wants to allow this and give the programmer reasonable behavior, the *longjmp()* function has to
 32374 call all cancellation cleanup handlers that have been pushed but not popped since the time
 32375 *setjmp()* was called.

32376 Consider a multi-threaded function called by a thread that uses signals. If a signal were
 32377 delivered to a signal handler during the operation of *qsort()* and that handler were to call
 32378 *longjmp()* (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads
 32379 created by the *qsort()* function would not be canceled. Instead, they would continue to execute
 32380 and write into the argument array even though the array might have been popped off the stack.

32381 Note that the specified cleanup handling mechanism is especially tied to the C language and,
 32382 while the requirement for a uniform mechanism for expressing cleanup is language-
 32383 independent, the mechanism used in other languages may be quite different. In addition, this
 32384 mechanism is really only necessary due to the lack of a real exception mechanism in the C
 32385 language, which would be the ideal solution.

32386 There is no notion of a cancellation cleanup-safe function. If an application has no cancellation
 32387 points in its signal handlers, blocks any signal whose handler may have cancellation points
 32388 while calling async-unsafe functions, or disables cancellation while calling async-unsafe
 32389 functions, all functions may be safely called from cancellation cleanup routines.

32390 **FUTURE DIRECTIONS**

32391 None.

32392 **SEE ALSO**

32393 *pthread_cancel()*, *pthread_setcancelstate()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 32394 **<pthread.h>**

32395 **CHANGE HISTORY**

32396 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32397 **Issue 6**

32398 The *pthread_cleanup_pop()* and *pthread_cleanup_push()* functions are marked as part of the
32399 Threads option.

32400 The APPLICATION USAGE section is added.

32401 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

32402 NAME

32403 pthread_cond_broadcast, pthread_cond_signal — broadcast or signal a condition

32404 SYNOPSIS

32405 THR `#include <pthread.h>`

32406 `int pthread_cond_broadcast(pthread_cond_t *cond);`

32407 `int pthread_cond_signal(pthread_cond_t *cond);`

32408

32409 DESCRIPTION

32410 These functions shall unblock threads blocked on a condition variable.

32411 The *pthread_cond_broadcast()* function shall unblock all threads currently blocked on the
32412 specified condition variable *cond*.

32413 The *pthread_cond_signal()* function shall unblock at least one of the threads that are blocked on
32414 the specified condition variable *cond* (if any threads are blocked on *cond*).

32415 If more than one thread is blocked on a condition variable, the scheduling policy shall determine
32416 the order in which threads are unblocked. When each thread unblocked as a result of a
32417 *pthread_cond_broadcast()* or *pthread_cond_signal()* returns from its call to *pthread_cond_wait()* or
32418 *pthread_cond_timedwait()*, the thread shall own the mutex with which it called
32419 *pthread_cond_wait()* or *pthread_cond_timedwait()*. The thread(s) that are unblocked shall contend
32420 for the mutex according to the scheduling policy (if applicable), and as if each had called
32421 *pthread_mutex_lock()*.

32422 The *pthread_cond_broadcast()* or *pthread_cond_signal()* functions may be called by a thread
32423 whether or not it currently owns the mutex that threads calling *pthread_cond_wait()* or
32424 *pthread_cond_timedwait()* have associated with the condition variable during their waits;
32425 however, if predictable scheduling behavior is required, then that mutex shall be locked by the
32426 thread calling *pthread_cond_broadcast()* or *pthread_cond_signal()*.

32427 The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions shall have no effect if there are
32428 no threads currently blocked on *cond*.

32429 RETURN VALUE

32430 If successful, the *pthread_cond_broadcast()* and *pthread_cond_signal()* functions shall return zero;
32431 otherwise, an error number shall be returned to indicate the error.

32432 ERRORS

32433 The *pthread_cond_broadcast()* and *pthread_cond_signal()* function may fail if:

32434 [EINVAL] The value *cond* does not refer to an initialized condition variable.

32435 These functions shall not return an error code of [EINTR].

32436 EXAMPLES

32437 None.

32438 APPLICATION USAGE

32439 The *pthread_cond_broadcast()* function is used whenever the shared-variable state has been
32440 changed in a way that more than one thread can proceed with its task. Consider a single
32441 producer/multiple consumer problem, where the producer can insert multiple items on a list
32442 that is accessed one item at a time by the consumers. By calling the *pthread_cond_broadcast()*
32443 function, the producer would notify all consumers that might be waiting, and thereby the
32444 application would receive more throughput on a multi-processor. In addition,
32445 *pthread_cond_broadcast()* makes it easier to implement a read-write lock. The
32446 *pthread_cond_broadcast()* function is needed in order to wake up all waiting readers when a

writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function to notify all clients of an impending transaction commit.

It is not safe to use the *pthread_cond_signal()* function in a signal handler that is invoked asynchronously. Even if it were safe, there would still be a race between the test of the Boolean *pthread_cond_wait()* that could not be efficiently eliminated.

Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling from code running in a signal handler.

RATIONALE

Multiple Awakenings by Condition Signal

On a multi-processor, it may be impossible for an implementation of *pthread_cond_signal()* to avoid the unblocking of more than one thread blocked on a condition variable. For example, consider the following partial implementation of *pthread_cond_wait()* and *pthread_cond_signal()*, executed by two threads in the order given. One thread is trying to wait on the condition variable, another is concurrently executing *pthread_cond_signal()*, while a third thread is already waiting.

```
pthread_cond_wait(mutex, cond):
    value = cond->value; /* 1 */
    pthread_mutex_unlock(mutex); /* 2 */
    pthread_mutex_lock(cond->mutex); /* 10 */
    if (value == cond->value) { /* 11 */
        me->next_cond = cond->waiter;
        cond->waiter = me;
        pthread_mutex_unlock(cond->mutex);
        unable_to_run(me);
    } else
        pthread_mutex_unlock(cond->mutex); /* 12 */
    pthread_mutex_lock(mutex); /* 13 */

pthread_cond_signal(cond):
    pthread_mutex_lock(cond->mutex); /* 3 */
    cond->value++; /* 4 */
    if (cond->waiter) { /* 5 */
        sleeper = cond->waiter; /* 6 */
        cond->waiter = sleeper->next_cond; /* 7 */
        able_to_run(sleeper); /* 8 */
    }
    pthread_mutex_unlock(cond->mutex); /* 9 */
```

The effect is that more than one thread can return from its call to *pthread_cond_wait()* or *pthread_cond_timedwait()* as a result of one call to *pthread_cond_signal()*. This effect is called “spurious wakeup”. Note that the situation is self-correcting in that the number of threads that are so awakened is finite; for example, the next thread to call *pthread_cond_wait()* after the sequence of events above blocks.

While this problem could be resolved, the loss of efficiency for a fringe condition that occurs only rarely is unacceptable, especially given that one has to check the predicate associated with a condition variable anyway. Correcting this problem would unnecessarily reduce the degree of concurrency in this basic building block for all higher-level synchronization operations.

An added benefit of allowing spurious wakeups is that applications are forced to code a predicate-testing-loop around the condition wait. This also makes the application tolerate

32494 superfluous condition broadcasts or signals on the same condition variable that may be coded in
32495 some other part of the application. The resulting applications are thus more robust. Therefore,
32496 IEEE Std 1003.1-2001 explicitly documents that spurious wakeups may occur.

32497 **FUTURE DIRECTIONS**

32498 None.

32499 **SEE ALSO**

32500 *pthread_cond_destroy()*, *pthread_cond_timedwait()*, the Base Definitions volume of
32501 IEEE Std 1003.1-2001, <**pthread.h**>

32502 **CHANGE HISTORY**

32503 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32504 **Issue 6**

32505 The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions are marked as part of the
32506 Threads option.

32507 The APPLICATION USAGE section is added.

32508 **NAME**

32509 pthread_cond_destroy, pthread_cond_init — destroy and initialize condition variables

32510 **SYNOPSIS**

32511 THR #include <pthread.h>

```

32512 int pthread_cond_destroy(pthread_cond_t *cond);
32513 int pthread_cond_init(pthread_cond_t *restrict cond,
32514     const pthread_condattr_t *restrict attr);
32515 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
32516

```

32517 **DESCRIPTION**

32518 The *pthread_cond_destroy()* function shall destroy the given condition variable specified by *cond*;
 32519 the object becomes, in effect, uninitialized. An implementation may cause *pthread_cond_destroy()*
 32520 to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can
 32521 be reinitialized using *pthread_cond_init()*; the results of otherwise referencing the object after it
 32522 has been destroyed are undefined.

32523 It shall be safe to destroy an initialized condition variable upon which no threads are currently
 32524 blocked. Attempting to destroy a condition variable upon which other threads are currently
 32525 blocked results in undefined behavior.

32526 The *pthread_cond_init()* function shall initialize the condition variable referenced by *cond* with
 32527 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be
 32528 used; the effect is the same as passing the address of a default condition variable attributes
 32529 object. Upon successful initialization, the state of the condition variable shall become initialized.

32530 Only *cond* itself may be used for performing synchronization. The result of referring to copies of
 32531 *cond* in calls to *pthread_cond_wait()*, *pthread_cond_timedwait()*, *pthread_cond_signal()*,
 32532 *pthread_cond_broadcast()*, and *pthread_cond_destroy()* is undefined.

32533 Attempting to initialize an already initialized condition variable results in undefined behavior.

32534 In cases where default condition variable attributes are appropriate, the macro
 32535 PTHREAD_COND_INITIALIZER can be used to initialize condition variables that are statically
 32536 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread_cond_init()*
 32537 with parameter *attr* specified as NULL, except that no error checks are performed.

32538 **RETURN VALUE**

32539 If successful, the *pthread_cond_destroy()* and *pthread_cond_init()* functions shall return zero;
 32540 otherwise, an error number shall be returned to indicate the error.

32541 The [EBUSY] and [EINVAL] error checks, if implemented, shall act as if they were performed
 32542 immediately at the beginning of processing for the function and caused an error return prior to
 32543 modifying the state of the condition variable specified by *cond*.

32544 **ERRORS**

32545 The *pthread_cond_destroy()* function may fail if:

32546 [EBUSY] The implementation has detected an attempt to destroy the object referenced
 32547 by *cond* while it is referenced (for example, while being used in a
 32548 *pthread_cond_wait()* or *pthread_cond_timedwait()*) by another thread.

32549 [EINVAL] The value specified by *cond* is invalid.

32550 The *pthread_cond_init()* function shall fail if:

32551 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
 32552 another condition variable.

32553 [ENOMEM] Insufficient memory exists to initialize the condition variable.

32554 The `pthread_cond_init()` function may fail if:

32555 [EBUSY] The implementation has detected an attempt to reinitialize the object
 32556 referenced by `cond`, a previously initialized, but not yet destroyed, condition
 32557 variable.

32558 [EINVAL] The value specified by `attr` is invalid.

32559 These functions shall not return an error code of [EINTR].

EXAMPLES

32561 A condition variable can be destroyed immediately after all the threads that are blocked on it are
 32562 awakened. For example, consider the following code:

```

32563 struct list {
32564     pthread_mutex_t lm;
32565     ...
32566 }

32567 struct elt {
32568     key k;
32569     int busy;
32570     pthread_cond_t notbusy;
32571     ...
32572 }

32573 /* Find a list element and reserve it. */
32574 struct elt *
32575 list_find(struct list *lp, key k)
32576 {
32577     struct elt *ep;

32578     pthread_mutex_lock(&lp->lm);
32579     while ((ep = find_elt(l, k) != NULL) && ep->busy)
32580         pthread_cond_wait(&ep->notbusy, &lp->lm);
32581     if (ep != NULL)
32582         ep->busy = 1;
32583     pthread_mutex_unlock(&lp->lm);
32584     return(ep);
32585 }

32586 delete_elt(struct list *lp, struct elt *ep)
32587 {
32588     pthread_mutex_lock(&lp->lm);
32589     assert(ep->busy);
32590     ... remove ep from list ...
32591     ep->busy = 0; /* Paranoid. */
32592     (A) pthread_cond_broadcast(&ep->notbusy);
32593     pthread_mutex_unlock(&lp->lm);
32594     (B) pthread_cond_destroy(&ep->notbusy);
32595     free(ep);
32596 }

```

32597 In this example, the condition variable and its list element may be freed (line B) immediately
 32598 after all threads waiting for it are awakened (line A), since the mutex and the code ensure that no
 32599 other thread can touch the element to be deleted.

32600 APPLICATION USAGE

32601 None.

32602 RATIONALE

32603 See *pthread_mutex_init()*; a similar rationale applies to condition variables.

32604 FUTURE DIRECTIONS

32605 None.

32606 SEE ALSO

32607 *pthread_cond_broadcast()*, *pthread_cond_signal()*, *pthread_cond_timedwait()*, the Base Definitions
32608 volume of IEEE Std 1003.1-2001, <**pthread.h**>

32609 CHANGE HISTORY

32610 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32611 Issue 6

32612 The *pthread_cond_destroy()* and *pthread_cond_init()* functions are marked as part of the Threads
32613 option.

32614 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

32615 The **restrict** keyword is added to the *pthread_cond_init()* prototype for alignment with the
32616 ISO / IEC 9899:1999 standard.

32617 **NAME**

32618 pthread_cond_signal — signal a condition

32619 **SYNOPSIS**32620 THR `#include <pthread.h>`32621 `int pthread_cond_signal(pthread_cond_t *cond);`

32622

32623 **DESCRIPTION**32624 Refer to *pthread_cond_broadcast()*.

32625 **NAME**

32626 pthread_cond_timedwait, pthread_cond_wait — wait on a condition

32627 **SYNOPSIS**

32628 THR #include <pthread.h>

```

32629 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
32630 pthread_mutex_t *restrict mutex,
32631 const struct timespec *restrict abstime);
32632 int pthread_cond_wait(pthread_cond_t *restrict cond,
32633 pthread_mutex_t *restrict mutex);
32634

```

32635 **DESCRIPTION**

32636 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions shall block on a condition
 32637 variable. They shall be called with *mutex* locked by the calling thread or undefined behavior
 32638 results.

32639 These functions atomically release *mutex* and cause the calling thread to block on the condition
 32640 variable *cond*; atomically here means “atomically with respect to access by another thread to the
 32641 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex
 32642 after the about-to-block thread has released it, then a subsequent call to *pthread_cond_broadcast()*
 32643 or *pthread_cond_signal()* in that thread shall behave as if it were issued after the about-to-block
 32644 thread has blocked.

32645 Upon successful return, the mutex shall have been locked and shall be owned by the calling
 32646 thread.

32647 When using condition variables there is always a Boolean predicate involving shared variables
 32648 associated with each condition wait that is true if the thread should proceed. Spurious wakeups
 32649 from the *pthread_cond_timedwait()* or *pthread_cond_wait()* functions may occur. Since the return
 32650 from *pthread_cond_timedwait()* or *pthread_cond_wait()* does not imply anything about the value
 32651 of this predicate, the predicate should be re-evaluated upon such return.

32652 The effect of using more than one mutex for concurrent *pthread_cond_timedwait()* or
 32653 *pthread_cond_wait()* operations on the same condition variable is undefined; that is, a condition
 32654 variable becomes bound to a unique mutex when a thread waits on the condition variable, and
 32655 this (dynamic) binding shall end when the wait returns.

32656 A condition wait (whether timed or not) is a cancellation point. When the cancelability enable
 32657 state of a thread is set to *PTHREAD_CANCEL_DEFERRED*, a side effect of acting upon a
 32658 cancellation request while in a condition wait is that the mutex is (in effect) re-acquired before
 32659 calling the first cancellation cleanup handler. The effect is as if the thread were unblocked,
 32660 allowed to execute up to the point of returning from the call to *pthread_cond_timedwait()* or
 32661 *pthread_cond_wait()*, but at that point notices the cancellation request and instead of returning to
 32662 the caller of *pthread_cond_timedwait()* or *pthread_cond_wait()*, starts the thread cancellation
 32663 activities, which includes calling cancellation cleanup handlers.

32664 A thread that has been unblocked because it has been canceled while blocked in a call to
 32665 *pthread_cond_timedwait()* or *pthread_cond_wait()* shall not consume any condition signal that
 32666 may be directed concurrently at the condition variable if there are other threads blocked on the
 32667 condition variable.

32668 The *pthread_cond_timedwait()* function shall be equivalent to *pthread_cond_wait()*, except that an
 32669 error is returned if the absolute time specified by *abstime* passes (that is, system time equals or
 32670 exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time
 32671 specified by *abstime* has already been passed at the time of the call.

32672 cs If the Clock Selection option is supported, the condition variable shall have a clock attribute
 32673 which specifies the clock that shall be used to measure the time specified by the *abstime*
 32674 argument. When such timeouts occur, *pthread_cond_timedwait()* shall nonetheless release and
 32675 re-acquire the mutex referenced by *mutex*. The *pthread_cond_timedwait()* function is also a
 32676 cancellation point.

32677 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal
 32678 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it
 32679 shall return zero due to spurious wakeup.

32680 RETURN VALUE

32681 Except in the case of [ETIMEDOUT], all these error checks shall act as if they were performed
 32682 immediately at the beginning of processing for the function and shall cause an error return, in
 32683 effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable
 32684 specified by *cond*.

32685 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall
 32686 be returned to indicate the error.

32687 ERRORS

32688 The *pthread_cond_timedwait()* function shall fail if:

32689 [ETIMEDOUT] The time specified by *abstime* to *pthread_cond_timedwait()* has passed.

32690 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions may fail if:

32691 [EINVAL] The value specified by *cond*, *mutex*, or *abstime* is invalid.

32692 [EINVAL] Different mutexes were supplied for concurrent *pthread_cond_timedwait()* or
 32693 *pthread_cond_wait()* operations on the same condition variable.

32694 [EPERM] The mutex was not owned by the current thread at the time of the call.

32695 These functions shall not return an error code of [EINTR].

32696 EXAMPLES

32697 None.

32698 APPLICATION USAGE

32699 None.

32700 RATIONALE

32701 Condition Wait Semantics

32702 It is important to note that when *pthread_cond_wait()* and *pthread_cond_timedwait()* return
 32703 without error, the associated predicate may still be false. Similarly, when
 32704 *pthread_cond_timedwait()* returns with the timeout error, the associated predicate may be true
 32705 due to an unavoidable race between the expiration of the timeout and the predicate state change.

32706 Some implementations, particularly on a multi-processor, may sometimes cause multiple
 32707 threads to wake up when the condition variable is signaled simultaneously on different
 32708 processors.

32709 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate
 32710 associated with the condition wait to determine whether it can safely proceed, should wait
 32711 again, or should declare a timeout. A return from the wait does not imply that the associated
 32712 predicate is either true or false.

32713 It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop”
 32714 that checks the predicate.

Timed Wait Semantics

An absolute time measure was chosen for specifying the timeout parameter for two reasons. First, a relative time measure can be easily implemented on top of a function that specifies absolute time, but there is a race condition associated with specifying an absolute timeout on top of a function that specifies relative timeouts. For example, assume that `clock_gettime()` returns the current time and `cond_relative_timed_wait()` uses relative timeouts:

```
clock_gettime(CLOCK_REALTIME, &now)
reltime = sleep_til_this_absolute_time - now;
cond_relative_timed_wait(c, m, &reltime);
```

If the thread is preempted between the first statement and the last statement, the thread blocks for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout also need not be recomputed if it is used multiple times in a loop, such as that enclosing a condition wait.

For cases when the system clock is advanced discontinuously by an operator, it is expected that implementations process any timed wait expiring at an intervening time as if that time had actually occurred.

Cancellation and Condition Wait

A condition wait, whether timed or not, is a cancellation point. That is, the functions `pthread_cond_wait()` or `pthread_cond_timedwait()` are points where a pending (or concurrent) cancellation request is noticed. The reason for this is that an indefinite wait is possible at these points—whatever event is being waited for, even if the program is totally correct, might never occur; for example, some input data being awaited might never be sent. By making condition wait a cancellation point, the thread can be canceled and perform its cancellation cleanup handler even though it may be stuck in some indefinite wait.

A side effect of acting on a cancellation request while a thread is blocked on a condition variable is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in order to ensure that the cancellation cleanup handler is executed in the same state as the critical code that lies both before and after the call to the condition wait function. This rule is also required when interfacing to POSIX threads from languages, such as Ada or C++, which may choose to map cancellation onto a language exception; this rule ensures that each exception handler guarding a critical section can always safely depend upon the fact that the associated mutex has already been locked regardless of exactly where within the critical section the exception was raised. Without this rule, there would not be a uniform rule that exception handlers could follow regarding the lock, and so coding would become very cumbersome.

Therefore, since *some* statement has to be made regarding the state of the lock when a cancellation is delivered during a wait, a definition has been chosen that makes application coding most convenient and error free.

When acting on a cancellation request while a thread is blocked on a condition variable, the implementation is required to ensure that the thread does not consume any condition signals directed at that condition variable if there are any other threads waiting on that condition variable. This rule is specified in order to avoid deadlock conditions that could occur if these two independent requests (one acting on a thread and the other acting on the condition variable) were not processed independently.

Performance of Mutexes and Condition Variables

Mutexes are expected to be locked only for a few instructions. This practice is almost automatically enforced by the desire of programmers to avoid long serial regions of execution (which would reduce total effective parallelism).

When using mutexes and condition variables, one tries to ensure that the usual case is to lock the mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a relatively rare situation. For example, when implementing a read-write lock, code that acquires a read-lock typically needs only to increment the count of readers (under mutual-exclusion) and return. The calling thread would actually wait on the condition variable only when there is already an active writer. So the efficiency of a synchronization operation is bounded by the cost of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context switch.

This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable is important. The cost of waiting on a condition variable should be little more than the minimal cost for a context switch plus the time to unlock and lock the mutex.

Features of Mutexes and Condition Variables

It had been suggested that the mutex acquisition and release be decoupled from condition wait. This was rejected because it is the combined nature of the operation that, in fact, facilitates realtime implementations. Those implementations can atomically move a high-priority thread between the condition variable and the mutex in a manner that is transparent to the caller. This can prevent extra context switches and provide more deterministic acquisition of a mutex when the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the scheduling discipline. Furthermore, the current condition wait operation matches existing practice.

Scheduling Behavior of Mutexes and Condition Variables

Synchronization primitives that attempt to interfere with scheduling policy by specifying an ordering rule are considered undesirable. Threads waiting on mutexes and condition variables are selected to proceed in an order dependent upon the scheduling policy rather than in some fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which thread(s) are awakened and allowed to proceed.

Timed Condition Wait

The `pthread_cond_timedwait()` function allows an application to give up waiting for a particular condition after a given amount of time. An example of its use follows:

```
(void) pthread_mutex_lock(&t.mn);
        t.waiters++;
        clock_gettime(CLOCK_REALTIME, &ts);
        ts.tv_sec += 5;
        rc = 0;
        while (! mypredicate(&t) && rc == 0)
            rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
        t.waiters--;
        if (rc == 0) setmystate(&t);
(void) pthread_mutex_unlock(&t.mn);
```

32802 By making the timeout parameter absolute, it does not need to be recomputed each time the
32803 program checks its blocking predicate. If the timeout was relative, it would have to be
32804 recomputed before each call. This would be especially difficult since such code would need to
32805 take into account the possibility of extra wakeups that result from extra broadcasts or signals on
32806 the condition variable that occur before either the predicate is true or the timeout is due.

32807 FUTURE DIRECTIONS

32808 None.

32809 SEE ALSO

32810 *pthread_cond_signal()*, *pthread_cond_broadcast()*, the Base Definitions volume of
32811 IEEE Std 1003.1-2001, <pthread.h>

32812 CHANGE HISTORY

32813 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32814 Issue 6

32815 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions are marked as part of the
32816 Threads option.

32817 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the
32818 *pthread_cond_wait()* function.

32819 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for
32820 the Clock Selection option.

32821 The ERRORS section has an additional case for [EPERM] in response to IEEE PASC
32822 Interpretation 1003.1c #28.

32823 The **restrict** keyword is added to the *pthread_cond_timedwait()* and *pthread_cond_wait()*
32824 prototypes for alignment with the ISO / IEC 9899:1999 standard.

32825 NAME

32826 pthread_condattr_destroy, pthread_condattr_init — destroy and initialize the condition variable
 32827 attributes object

32828 SYNOPSIS

32829 THR `#include <pthread.h>`

32830 `int pthread_condattr_destroy(pthread_condattr_t *attr);`

32831 `int pthread_condattr_init(pthread_condattr_t *attr);`

32832

32833 DESCRIPTION

32834 The *pthread_condattr_destroy()* function shall destroy a condition variable attributes object; the
 32835 object becomes, in effect, uninitialized. An implementation may cause *pthread_condattr_destroy()*
 32836 to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be
 32837 reinitialized using *pthread_condattr_init()*; the results of otherwise referencing the object after it
 32838 has been destroyed are undefined.

32839 The *pthread_condattr_init()* function shall initialize a condition variable attributes object *attr* with
 32840 the default value for all of the attributes defined by the implementation.

32841 Results are undefined if *pthread_condattr_init()* is called specifying an already initialized *attr*
 32842 attributes object.

32843 After a condition variable attributes object has been used to initialize one or more condition
 32844 variables, any function affecting the attributes object (including destruction) shall not affect any
 32845 previously initialized condition variables.

32846 This volume of IEEE Std 1003.1-2001 requires two attributes, the *clock* attribute and the *process-*
 32847 *shared* attribute.

32848 Additional attributes, their default values, and the names of the associated functions to get and
 32849 set those attribute values are implementation-defined.

32850 RETURN VALUE

32851 If successful, the *pthread_condattr_destroy()* and *pthread_condattr_init()* functions shall return
 32852 zero; otherwise, an error number shall be returned to indicate the error.

32853 ERRORS

32854 The *pthread_condattr_destroy()* function may fail if:

32855 [EINVAL] The value specified by *attr* is invalid.

32856 The *pthread_condattr_init()* function shall fail if:

32857 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

32858 These functions shall not return an error code of [EINTR].

32859 EXAMPLES

32860 None.

32861 APPLICATION USAGE

32862 None.

32863 RATIONALE

32864 See *pthread_attr_init()* and *pthread_mutex_init()*.

32865 A *process-shared* attribute has been defined for condition variables for the same reason it has been
 32866 defined for mutexes.

32867 FUTURE DIRECTIONS

32868 None.

32869 SEE ALSO

32870 *pthread_attr_destroy()*, *pthread_cond_destroy()*, *pthread_condattr_getpshared()*, *pthread_create()*,
32871 *pthread_mutex_destroy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

32872 CHANGE HISTORY

32873 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32874 Issue 6

32875 The *pthread_condattr_destroy()* and *pthread_condattr_init()* functions are marked as part of the
32876 Threads option.

32877 **NAME**

32878 pthread_condattr_getclock, pthread_condattr_setclock — get and set the clock selection
 32879 condition variable attribute (**ADVANCED REALTIME**)

32880 **SYNOPSIS**

```
32881 THR CS #include <pthread.h>

32882 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
32883 clockid_t *restrict clock_id);
32884 int pthread_condattr_setclock(pthread_condattr_t *attr,
32885 clockid_t clock_id);
32886
```

32887 **DESCRIPTION**

32888 The *pthread_condattr_getclock()* function shall obtain the value of the *clock* attribute from the
 32889 attributes object referenced by *attr*. The *pthread_condattr_setclock()* function shall set the *clock*
 32890 attribute in an initialized attributes object referenced by *attr*. If *pthread_condattr_setclock()* is
 32891 called with a *clock_id* argument that refers to a CPU-time clock, the call shall fail.

32892 The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of
 32893 *pthread_cond_timedwait()*. The default value of the *clock* attribute shall refer to the system clock.

32894 **RETURN VALUE**

32895 If successful, the *pthread_condattr_getclock()* function shall return zero and store the value of the
 32896 clock attribute of *attr* into the object referenced by the *clock_id* argument. Otherwise, an error
 32897 number shall be returned to indicate the error.

32898 If successful, the *pthread_condattr_setclock()* function shall return zero; otherwise, an error
 32899 number shall be returned to indicate the error.

32900 **ERRORS**

32901 These functions may fail if:

32902 [EINVAL] The value specified by *attr* is invalid.

32903 The *pthread_condattr_setclock()* function may fail if:

32904 [EINVAL] The value specified by *clock_id* does not refer to a known clock, or is a CPU-
 32905 time clock.

32906 These functions shall not return an error code of [EINTR].

32907 **EXAMPLES**

32908 None.

32909 **APPLICATION USAGE**

32910 None.

32911 **RATIONALE**

32912 None.

32913 **FUTURE DIRECTIONS**

32914 None.

32915 **SEE ALSO**

32916 *pthread_cond_destroy()*, *pthread_cond_timedwait()*, *pthread_condattr_destroy()*,
 32917 *pthread_condattr_getpshared()* (on page 1045), *pthread_condattr_init()*,
 32918 *pthread_condattr_setpshared()* (on page 1049), *pthread_create()*, *pthread_mutex_init()*, the Base
 32919 Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

32920 **CHANGE HISTORY**

32921 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32922 NAME

32923 pthread_condattr_getpshared, pthread_condattr_setpshared — get and set the process-shared
 32924 condition variable attributes

32925 SYNOPSIS

```
32926 THR TSH #include <pthread.h>

32927 int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
32928 int *restrict pshared);
32929 int pthread_condattr_setpshared(pthread_condattr_t *attr,
32930 int pshared);
32931
```

32932 DESCRIPTION

32933 The *pthread_condattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 32934 from the attributes object referenced by *attr*. The *pthread_condattr_setpshared()* function shall set
 32935 the *process-shared* attribute in an initialized attributes object referenced by *attr*.

32936 The *process-shared* attribute is set to *PTHREAD_PROCESS_SHARED* to permit a condition
 32937 variable to be operated upon by any thread that has access to the memory where the condition
 32938 variable is allocated, even if the condition variable is allocated in memory that is shared by
 32939 multiple processes. If the *process-shared* attribute is *PTHREAD_PROCESS_PRIVATE*, the
 32940 condition variable shall only be operated upon by threads created within the same process as the
 32941 thread that initialized the condition variable; if threads of differing processes attempt to operate
 32942 on such a condition variable, the behavior is undefined. The default value of the attribute is
 32943 *PTHREAD_PROCESS_PRIVATE*.

32944 RETURN VALUE

32945 If successful, the *pthread_condattr_setpshared()* function shall return zero; otherwise, an error
 32946 number shall be returned to indicate the error.

32947 If successful, the *pthread_condattr_getpshared()* function shall return zero and store the value of
 32948 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,
 32949 an error number shall be returned to indicate the error.

32950 ERRORS

32951 The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions may fail if:

32952 [EINVAL] The value specified by *attr* is invalid.

32953 The *pthread_condattr_setpshared()* function may fail if:

32954 [EINVAL] The new value specified for the attribute is outside the range of legal values
 32955 for that attribute.

32956 These functions shall not return an error code of [EINTR].

32957 EXAMPLES

32958 None.

32959 APPLICATION USAGE

32960 None.

32961 RATIONALE

32962 None.

32963 FUTURE DIRECTIONS

32964 None.

32965 SEE ALSO

32966 *pthread_create()*, *pthread_cond_destroy()*, *pthread_condattr_destroy()*, *pthread_mutex_destroy()*, the
32967 Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

32968 CHANGE HISTORY

32969 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32970 Issue 6

32971 The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions are marked as part
32972 of the Threads and Thread Process-Shared Synchronization options.

32973 The **restrict** keyword is added to the *pthread_condattr_getpshared()* prototype for alignment with
32974 the ISO / IEC 9899:1999 standard.

32975 **NAME**

32976 pthread_condattr_init — initialize the condition variable attributes object

32977 **SYNOPSIS**

32978 THR #include <pthread.h>

32979 int pthread_condattr_init(pthread_condattr_t *attr);

32980

32981 **DESCRIPTION**

32982 Refer to *pthread_condattr_destroy()*.

32983 NAME

32984 pthread_condattr_setclock — set the clock selection condition variable attribute

32985 SYNOPSIS

32986 THR CS #include <pthread.h>

```
32987 int pthread_condattr_setclock(pthread_condattr_t *attr,  
32988 clockid_t clock_id);
```

32989

32990 DESCRIPTION

32991 Refer to *pthread_condattr_getclock()*.

32992 **NAME**

32993 pthread_condattr_setpshared — set the process-shared condition variable attribute

32994 **SYNOPSIS**

32995 THR TSH `#include <pthread.h>`

32996 `int pthread_condattr_setpshared(pthread_condattr_t *attr,`
32997 `int pshared);`

32998

32999 **DESCRIPTION**

33000 Refer to *pthread_condattr_getpshared()*.

33001 **NAME**

33002 pthread_create — thread creation

33003 **SYNOPSIS**

33004 THR #include <pthread.h>

```
33005 int pthread_create(pthread_t *restrict thread,
33006     const pthread_attr_t *restrict attr,
33007     void *(*start_routine) (void*), void *restrict arg);
33008
```

33009 **DESCRIPTION**

33010 The *pthread_create()* function shall create a new thread, with attributes specified by *attr*, within a
 33011 process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are
 33012 modified later, the thread's attributes shall not be affected. Upon successful completion,
 33013 *pthread_create()* shall store the ID of the created thread in the location referenced by *thread*.

33014 The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine*
 33015 returns, the effect shall be as if there was an implicit call to *pthread_exit()* using the return value
 33016 of *start_routine* as the exit status. Note that the thread in which *main()* was originally invoked
 33017 differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call
 33018 to *exit()* using the return value of *main()* as the exit status.

33019 The signal state of the new thread shall be initialized as follows:

- 33020 • The signal mask shall be inherited from the creating thread.
- 33021 • The set of signals pending for the new thread shall be empty.

33022 XSI The alternate stack shall not be inherited.

33023 The floating-point environment shall be inherited from the creating thread.

33024 If *pthread_create()* fails, no new thread is created and the contents of the location referenced by
 33025 *thread* are undefined.

33026 TCT If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock
 33027 accessible, and the initial value of this clock shall be set to zero.

33028 **RETURN VALUE**

33029 If successful, the *pthread_create()* function shall return zero; otherwise, an error number shall be
 33030 returned to indicate the error.

33031 **ERRORS**

33032 The *pthread_create()* function shall fail if:

33033 [EAGAIN] The system lacked the necessary resources to create another thread, or the
 33034 system-imposed limit on the total number of threads in a process
 33035 {PTHREAD_THREADS_MAX} would be exceeded.

33036 [EINVAL] The value specified by *attr* is invalid.

33037 [EPERM] The caller does not have appropriate permission to set the required
 33038 scheduling parameters or scheduling policy.

33039 The *pthread_create()* function shall not return an error code of [EINTR].

33040 **EXAMPLES**

33041 None.

33042 **APPLICATION USAGE**

33043 None.

33044 **RATIONALE**

33045 A suggested alternative to *pthread_create()* would be to define two separate operations: create
 33046 and start. Some applications would find such behavior more natural. Ada, in particular,
 33047 separates the “creation” of a task from its “activation”.

33048 Splitting the operation was rejected by the standard developers for many reasons:

- 33049 • The number of calls required to start a thread would increase from one to two and thus place
 33050 an additional burden on applications that do not require the additional synchronization. The
 33051 second call, however, could be avoided by the additional complication of a start-up state
 33052 attribute.
- 33053 • An extra state would be introduced: “created but not started”. This would require the
 33054 standard to specify the behavior of the thread operations when the target has not yet started
 33055 executing.
- 33056 • For those applications that require such behavior, it is possible to simulate the two separate
 33057 steps with the facilities that are currently provided. The *start_routine()* can synchronize by
 33058 waiting on a condition variable that is signaled by the start operation.

33059 An Ada implementor can choose to create the thread at either of two points in the Ada program:
 33060 when the task object is created, or when the task is activated (generally at a “begin”). If the first
 33061 approach is adopted, the *start_routine()* needs to wait on a condition variable to receive the
 33062 order to begin “activation”. The second approach requires no such condition variable or extra
 33063 synchronization. In either approach, a separate Ada task control block would need to be created
 33064 when the task object is created to hold rendezvous queues, and so on.

33065 An extension of the preceding model would be to allow the state of the thread to be modified
 33066 between the create and start. This would allow the thread attributes object to be eliminated. This
 33067 has been rejected because:

- 33068 • All state in the thread attributes object has to be able to be set for the thread. This would
 33069 require the definition of functions to modify thread attributes. There would be no reduction
 33070 in the number of function calls required to set up the thread. In fact, for an application that
 33071 creates all threads using identical attributes, the number of function calls required to set up
 33072 the threads would be dramatically increased. Use of a thread attributes object permits the
 33073 application to make one set of attribute setting function calls. Otherwise, the set of attribute
 33074 setting function calls needs to be made for each thread creation.
- 33075 • Depending on the implementation architecture, functions to set thread state would require
 33076 kernel calls, or for other implementation reasons would not be able to be implemented as
 33077 macros, thereby increasing the cost of thread creation.
- 33078 • The ability for applications to segregate threads by class would be lost.

33079 Another suggested alternative uses a model similar to that for process creation, such as “thread
 33080 fork”. The fork semantics would provide more flexibility and the “create” function can be
 33081 implemented simply by doing a thread fork followed immediately by a call to the desired “start
 33082 routine” for the thread. This alternative has these problems:

- 33083 • For many implementations, the entire stack of the calling thread would need to be
 33084 duplicated, since in many architectures there is no way to determine the size of the calling
 33085 frame.

33086 • Efficiency is reduced since at least some part of the stack has to be copied, even though in
 33087 most cases the thread never needs the copied context, since it merely calls the desired start
 33088 routine.

33089 FUTURE DIRECTIONS

33090 None.

33091 SEE ALSO

33092 *fork()*, *pthread_exit()*, *pthread_join()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 33093 <pthread.h>

33094 CHANGE HISTORY

33095 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33096 Issue 6

33097 The *pthread_create()* function is marked as part of the Threads option.

33098 The following new requirements on POSIX implementations derive from alignment with the
 33099 Single UNIX Specification:

33100 • The [EPERM] mandatory error condition is added.

33101 The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

33102 The **restrict** keyword is added to the *pthread_create()* prototype for alignment with the
 33103 ISO/IEC 9899:1999 standard.

33104 The DESCRIPTION is updated to make it explicit that the floating-point environment is
 33105 inherited from the creating thread.

33106 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the
 33107 alternate stack is not inherited.

33108 **NAME**

33109 pthread_detach — detach a thread

33110 **SYNOPSIS**

33111 THR #include <pthread.h>

33112 int pthread_detach(pthread_t thread);

33113

33114 **DESCRIPTION**

33115 The *pthread_detach()* function shall indicate to the implementation that storage for the thread
 33116 *thread* can be reclaimed when that thread terminates. If *thread* has not terminated,
 33117 *pthread_detach()* shall not cause it to terminate. The effect of multiple *pthread_detach()* calls on
 33118 the same target thread is unspecified.

33119 **RETURN VALUE**

33120 If the call succeeds, *pthread_detach()* shall return 0; otherwise, an error number shall be returned
 33121 to indicate the error.

33122 **ERRORS**33123 The *pthread_detach()* function shall fail if:

33124 [EINVAL] The implementation has detected that the value specified by *thread* does not
 33125 refer to a joinable thread.

33126 [ESRCH] No thread could be found corresponding to that specified by the given thread
 33127 ID.

33128 The *pthread_detach()* function shall not return an error code of [EINTR].33129 **EXAMPLES**

33130 None.

33131 **APPLICATION USAGE**

33132 None.

33133 **RATIONALE**

33134 The *pthread_join()* or *pthread_detach()* functions should eventually be called for every thread that
 33135 is created so that storage associated with the thread may be reclaimed.

33136 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation
 33137 attribute is sufficient, since a thread need never be dynamically detached. However, need arises
 33138 in at least two cases:

33139 1. In a cancellation handler for a *pthread_join()* it is nearly essential to have a *pthread_detach()* |
 33140 function in order to detach the thread on which *pthread_join()* was waiting. Without it, it |
 33141 would be necessary to have the handler do another *pthread_join()* to attempt to detach the |
 33142 thread, which would both delay the cancellation processing for an unbounded period and |
 33143 introduce a new call to *pthread_join()*, which might itself need a cancellation handler. A |
 33144 dynamic detach is nearly essential in this case. |

33145 2. In order to detach the “initial thread” (as may be desirable in processes that set up server
 33146 threads).

33147 **FUTURE DIRECTIONS**

33148 None.

33149 **SEE ALSO**

33150 *pthread_join()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

33151 **CHANGE HISTORY**

33152 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33153 **Issue 6**

33154 The *pthread_detach()* function is marked as part of the Threads option.

33155 **NAME**

33156 pthread_equal — compare thread IDs

33157 **SYNOPSIS**

33158 THR #include <pthread.h>

33159 int pthread_equal(pthread_t t1, pthread_t t2);

33160

33161 **DESCRIPTION**33162 This function shall compare the thread IDs *t1* and *t2*.33163 **RETURN VALUE**33164 The *pthread_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero
33165 shall be returned.33166 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.33167 **ERRORS**

33168 No errors are defined.

33169 The *pthread_equal()* function shall not return an error code of [EINTR].33170 **EXAMPLES**

33171 None.

33172 **APPLICATION USAGE**

33173 None.

33174 **RATIONALE**33175 Implementations may choose to define a thread ID as a structure. This allows additional
33176 flexibility and robustness over using an **int**. For example, a thread ID could include a sequence
33177 number that allows detection of “dangling IDs” (copies of a thread ID that has been detached).
33178 Since the C language does not support comparison on structure types, the *pthread_equal()*
33179 function is provided to compare thread IDs.33180 **FUTURE DIRECTIONS**

33181 None.

33182 **SEE ALSO**33183 *pthread_create()*, *pthread_self()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>33184 **CHANGE HISTORY**

33185 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33186 **Issue 6**33187 The *pthread_equal()* function is marked as part of the Threads option.

33188 **NAME**

33189 pthread_exit — thread termination

33190 **SYNOPSIS**

33191 THR #include <pthread.h>

33192 void pthread_exit(void *value_ptr);

33193

33194 **DESCRIPTION**

33195 The *pthread_exit()* function shall terminate the calling thread and make the value *value_ptr*
 33196 available to any successful join with the terminating thread. Any cancellation cleanup handlers
 33197 that have been pushed and not yet popped shall be popped in the reverse order that they were
 33198 pushed and then executed. After all cancellation cleanup handlers have been executed, if the
 33199 thread has any thread-specific data, appropriate destructor functions shall be called in an
 33200 unspecified order. Thread termination does not release any application visible process resources,
 33201 including, but not limited to, mutexes and file descriptors, nor does it perform any process-level
 33202 cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

33203 An implicit call to *pthread_exit()* is made when a thread other than the thread in which *main()*
 33204 was first invoked returns from the start routine that was used to create it. The function's return
 33205 value shall serve as the thread's exit status.

33206 The behavior of *pthread_exit()* is undefined if called from a cancellation cleanup handler or
 33207 destructor function that was invoked as a result of either an implicit or explicit call to
 33208 *pthread_exit()*.

33209 After a thread has terminated, the result of access to local (auto) variables of the thread is
 33210 undefined. Thus, references to local variables of the exiting thread should not be used for the
 33211 *pthread_exit()* *value_ptr* parameter value.

33212 The process shall exit with an exit status of 0 after the last thread has been terminated. The
 33213 behavior shall be as if the implementation called *exit()* with a zero argument at thread
 33214 termination time.

33215 **RETURN VALUE**33216 The *pthread_exit()* function cannot return to its caller.33217 **ERRORS**

33218 No errors are defined.

33219 **EXAMPLES**

33220 None.

33221 **APPLICATION USAGE**

33222 None.

33223 **RATIONALE**

33224 The normal mechanism by which a thread terminates is to return from the routine that was
 33225 specified in the *pthread_create()* call that started it. The *pthread_exit()* function provides the
 33226 capability for a thread to terminate without requiring a return from the start routine of that
 33227 thread, thereby providing a function analogous to *exit()*.

33228 Regardless of the method of thread termination, any cancellation cleanup handlers that have
 33229 been pushed and not yet popped are executed, and the destructors for any existing thread-
 33230 specific data are executed. This volume of IEEE Std 1003.1-2001 requires that cancellation
 33231 cleanup handlers be popped and called in order. After all cancellation cleanup handlers have
 33232 been executed, thread-specific data destructors are called, in an unspecified order, for each item
 33233 of thread-specific data that exists in the thread. This ordering is necessary because cancellation

33234 cleanup handlers may rely on thread-specific data.

33235 As the meaning of the status is determined by the application (except when the thread has been
33236 canceled, in which case it is PTHREAD_CANCELED), the implementation has no idea what an
33237 illegal status value is, which is why no address error checking is done.

33238 **FUTURE DIRECTIONS**

33239 None.

33240 **SEE ALSO**

33241 *exit()*, *pthread_create()*, *pthread_join()*, the Base Definitions volume of IEEE Std 1003.1-2001,
33242 **<pthread.h>**

33243 **CHANGE HISTORY**

33244 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33245 **Issue 6**

33246 The *pthread_exit()* function is marked as part of the Threads option.

33247 **NAME**

33248 pthread_getconcurrency, pthread_setconcurrency — get and set the level of concurrency

33249 **SYNOPSIS**

33250 xSI #include <pthread.h>

33251 int pthread_getconcurrency(void);

33252 int pthread_setconcurrency(int new_level);

33253

33254 **DESCRIPTION**

33255 Unbound threads in a process may or may not be required to be simultaneously active. By
 33256 default, the threads implementation ensures that a sufficient number of threads are active so that
 33257 the process can continue to make progress. While this conserves system resources, it may not
 33258 produce the most effective level of concurrency.

33259 The *pthread_setconcurrency()* function allows an application to inform the threads
 33260 implementation of its desired concurrency level, *new_level*. The actual level of concurrency
 33261 provided by the implementation as a result of this function call is unspecified.

33262 If *new_level* is zero, it causes the implementation to maintain the concurrency level at its
 33263 discretion as if *pthread_setconcurrency()* had never been called.

33264 The *pthread_getconcurrency()* function shall return the value set by a previous call to the
 33265 *pthread_setconcurrency()* function. If the *pthread_setconcurrency()* function was not previously
 33266 called, this function shall return zero to indicate that the implementation is maintaining the
 33267 concurrency level.

33268 A call to *pthread_setconcurrency()* shall inform the implementation of its desired concurrency
 33269 level. The implementation shall use this as a hint, not a requirement.

33270 If an implementation does not support multiplexing of user threads on top of several kernel-
 33271 scheduled entities, the *pthread_setconcurrency()* and *pthread_getconcurrency()* functions are
 33272 provided for source code compatibility but they shall have no effect when called. To maintain
 33273 the function semantics, the *new_level* parameter is saved when *pthread_setconcurrency()* is called
 33274 so that a subsequent call to *pthread_getconcurrency()* shall return the same value.

33275 **RETURN VALUE**

33276 If successful, the *pthread_setconcurrency()* function shall return zero; otherwise, an error number
 33277 shall be returned to indicate the error.

33278 The *pthread_getconcurrency()* function shall always return the concurrency level set by a previous
 33279 call to *pthread_setconcurrency()*. If the *pthread_setconcurrency()* function has never been called,
 33280 *pthread_getconcurrency()* shall return zero.

33281 **ERRORS**33282 The *pthread_setconcurrency()* function shall fail if:33283 [EINVAL] The value specified by *new_level* is negative.33284 [EAGAIN] The value specific by *new_level* would cause a system resource to be exceeded.

33285 These functions shall not return an error code of [EINTR].

33286 EXAMPLES

33287 None.

33288 APPLICATION USAGE

33289 Use of these functions changes the state of the underlying concurrency upon which the
33290 application depends. Library developers are advised to not use the *pthread_getconcurrency()* and
33291 *pthread_setconcurrency()* functions since their use may conflict with an applications use of these
33292 functions.

33293 RATIONALE

33294 None.

33295 FUTURE DIRECTIONS

33296 None.

33297 SEE ALSO

33298 The Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

33299 CHANGE HISTORY

33300 First released in Issue 5.

33301 **NAME**

33302 pthread_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME**
 33303 **THREADS**)

33304 **SYNOPSIS**

33305 THR TCT #include <pthread.h>

33306 #include <time.h>

33307 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);

33308

33309 **DESCRIPTION**

33310 The *pthread_getcpuclockid()* function shall return in *clock_id* the clock ID of the CPU-time clock of
 33311 the thread specified by *thread_id*, if the thread specified by *thread_id* exists.

33312 **RETURN VALUE**

33313 Upon successful completion, *pthread_getcpuclockid()* shall return zero; otherwise, an error
 33314 number shall be returned to indicate the error.

33315 **ERRORS**

33316 The *pthread_getcpuclockid()* function may fail if:

33317 [ESRCH] The value specified by *thread_id* does not refer to an existing thread.

33318 **EXAMPLES**

33319 None.

33320 **APPLICATION USAGE**

33321 The *pthread_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not
 33322 be provided on all implementations.

33323 **RATIONALE**

33324 None.

33325 **FUTURE DIRECTIONS**

33326 None.

33327 **SEE ALSO**

33328 *clock_getcpuclockid()*, *clock_getres()*, *timer_create()*, the Base Definitions volume of
 33329 IEEE Std 1003.1-2001, <pthread.h>, <time.h>

33330 **CHANGE HISTORY**

33331 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

33332 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

33333 **NAME**

33334 pthread_getschedparam, pthread_setschedparam — dynamic thread scheduling parameters
 33335 access (**REALTIME THREADS**)

33336 **SYNOPSIS**

33337 THR TPS #include <pthread.h>

```
33338 int pthread_getschedparam(pthread_t thread, int *restrict policy,
33339 struct sched_param *restrict param);
33340 int pthread_setschedparam(pthread_t thread, int policy,
33341 const struct sched_param *param);
33342
```

33343 **DESCRIPTION**

33344 The *pthread_getschedparam()* and *pthread_setschedparam()* functions shall, respectively, get and set
 33345 the scheduling policy and parameters of individual threads within a multi-threaded process to
 33346 be retrieved and set. For SCHED_FIFO and SCHED_RR, the only required member of the
 33347 **sched_param** structure is the priority *sched_priority*. For SCHED_OTHER, the affected
 33348 scheduling parameters are implementation-defined.

33349 The *pthread_getschedparam()* function shall retrieve the scheduling policy and scheduling
 33350 parameters for the thread whose thread ID is given by *thread* and shall store those values in
 33351 *policy* and *param*, respectively. The priority value returned from *pthread_getschedparam()* shall be
 33352 the value specified by the most recent *pthread_setschedparam()*, *pthread_setschedprio()*, or
 33353 *pthread_create()* call affecting the target thread. It shall not reflect any temporary adjustments to
 33354 its priority as a result of any priority inheritance or ceiling functions. The *pthread_setschedparam()*
 33355 function shall set the scheduling policy and associated scheduling parameters for the thread
 33356 whose thread ID is given by *thread* to the policy and associated parameters provided in *policy*
 33357 and *param*, respectively.

33358 The *policy* parameter may have the value SCHED_OTHER, SCHED_FIFO, or SCHED_RR. The
 33359 scheduling parameters for the SCHED_OTHER policy are implementation-defined. The
 33360 SCHED_FIFO and SCHED_RR policies shall have a single scheduling parameter, *priority*.

33361 TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, then the *policy* argument may have the
 33362 value SCHED_SPORADIC, with the exception for the *pthread_setschedparam()* function that if the
 33363 scheduling policy was not SCHED_SPORADIC at the time of the call, it is implementation-
 33364 defined whether the function is supported; in other words, the implementation need not allow
 33365 the application to dynamically change the scheduling policy to SCHED_SPORADIC. The
 33366 sporadic server scheduling policy has the associated parameters *sched_ss_low_priority*,
 33367 *sched_ss_repl_period*, *sched_ss_init_budget*, *sched_priority*, and *sched_ss_max_repl*. The specified
 33368 *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the
 33369 function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall
 33370 be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function
 33371 shall fail.

33372 If the *pthread_setschedparam()* function fails, the scheduling parameters shall not be changed for
 33373 the target thread.

33374 **RETURN VALUE**

33375 If successful, the *pthread_getschedparam()* and *pthread_setschedparam()* functions shall return zero;
 33376 otherwise, an error number shall be returned to indicate the error.

33377 **ERRORS**

33378 The *pthread_getschedparam()* function may fail if:

33379 [ESRCH] The value specified by *thread* does not refer to an existing thread.

33380 The *pthread_setschedparam()* function may fail if:

33381 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated
33382 with the scheduling policy *policy* is invalid.

33383 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an
33384 unsupported value.

33385 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to
33386 SCHED_SPORADIC, and the implementation does not support this change.

33387 [EPERM] The caller does not have the appropriate permission to set either the
33388 scheduling parameters or the scheduling policy of the specified thread.

33389 [EPERM] The implementation does not allow the application to modify one of the
33390 parameters to the value specified.

33391 [ESRCH] The value specified by *thread* does not refer to a existing thread.

33392 These functions shall not return an error code of [EINTR].

33393 **EXAMPLES**

33394 None.

33395 **APPLICATION USAGE**

33396 None.

33397 **RATIONALE**

33398 None.

33399 **FUTURE DIRECTIONS**

33400 None.

33401 **SEE ALSO**

33402 *pthread_setschedprio()*, *sched_getparam()*, *sched_getscheduler()*, the Base Definitions volume of
33403 IEEE Std 1003.1-2001, <pthread.h>, <sched.h>

33404 **CHANGE HISTORY**

33405 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33406 **Issue 6**

33407 The *pthread_getschedparam()* and *pthread_setschedparam()* functions are marked as part of the
33408 Threads and Thread Execution Scheduling options.

33409 The [ENOSYS] error condition has been removed as stubs need not be provided if an
33410 implementation does not support the Thread Execution Scheduling option.

33411 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the
33412 *pthread_setschedparam()* function so that its second argument is of type **int**.

33413 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

33414 The **restrict** keyword is added to the *pthread_getschedparam()* prototype for alignment with the
33415 ISO/IEC 9899:1999 standard.

33416 The Open Group Corrigendum U047/1 is applied.

33417 IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a
33418 call to the *pthread_setschedprio()* function.

33419 **NAME**

33420 pthread_getspecific, pthread_setspecific — thread-specific data management

33421 **SYNOPSIS**

33422 THR #include <pthread.h>

33423 void *pthread_getspecific(pthread_key_t key);

33424 int pthread_setspecific(pthread_key_t key, const void *value);

33425

33426 **DESCRIPTION**33427 The *pthread_getspecific()* function shall return the value currently bound to the specified *key* on
33428 behalf of the calling thread.33429 The *pthread_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a
33430 previous call to *pthread_key_create()*. Different threads may bind different values to the same
33431 key. These values are typically pointers to blocks of dynamically allocated memory that have
33432 been reserved for use by the calling thread.33433 The effect of calling *pthread_getspecific()* or *pthread_setspecific()* with a *key* value not obtained
33434 from *pthread_key_create()* or after *key* has been deleted with *pthread_key_delete()* is undefined.33435 Both *pthread_getspecific()* and *pthread_setspecific()* may be called from a thread-specific data
33436 destructor function. A call to *pthread_getspecific()* for the thread-specific data *key* being
33437 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)
33438 by a call to *pthread_setspecific()*. Calling *pthread_setspecific()* from a thread-specific data
33439 destructor routine may result either in lost storage (after at least
33440 PTHREAD_DESTRUCTOR_ITERATIONS attempts at destruction) or in an infinite loop.

33441 Both functions may be implemented as macros.

33442 **RETURN VALUE**33443 The *pthread_getspecific()* function shall return the thread-specific data value associated with the
33444 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be
33445 returned.33446 If successful, the *pthread_setspecific()* function shall return zero; otherwise, an error number shall
33447 be returned to indicate the error.33448 **ERRORS**33449 No errors are returned from *pthread_getspecific()*.33450 The *pthread_setspecific()* function shall fail if:

33451 [ENOMEM] Insufficient memory exists to associate the value with the key.

33452 The *pthread_setspecific()* function may fail if:

33453 [EINVAL] The key value is invalid.

33454 These functions shall not return an error code of [EINTR].

33455 EXAMPLES

33456 None.

33457 APPLICATION USAGE

33458 None.

33459 RATIONALE

33460 Performance and ease-of-use of *pthread_getspecific()* are critical for functions that rely on
33461 maintaining state in thread-specific data. Since no errors are required to be detected by it, and
33462 since the only error that could be detected is the use of an invalid key, the function to
33463 *pthread_getspecific()* has been designed to favor speed and simplicity over error reporting.

33464 FUTURE DIRECTIONS

33465 None.

33466 SEE ALSO

33467 *pthread_key_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

33468 CHANGE HISTORY

33469 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33470 Issue 6

33471 The *pthread_getspecific()* and *pthread_setspecific()* functions are marked as part of the Threads
33472 option.

33473 IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

33474 **NAME**

33475 pthread_join — wait for thread termination

33476 **SYNOPSIS**

33477 THR #include <pthread.h>

33478 int pthread_join(pthread_t thread, void **value_ptr);

33479

33480 **DESCRIPTION**

33481 The *pthread_join()* function shall suspend execution of the calling thread until the target *thread*
 33482 terminates, unless the target *thread* has already terminated. On return from a successful
 33483 *pthread_join()* call with a non-NULL *value_ptr* argument, the value passed to *pthread_exit()* by
 33484 the terminating thread shall be made available in the location referenced by *value_ptr*. When a
 33485 *pthread_join()* returns successfully, the target thread has been terminated. The results of multiple
 33486 simultaneous calls to *pthread_join()* specifying the same target thread are undefined. If the
 33487 thread calling *pthread_join()* is canceled, then the target thread shall not be detached.

33488 It is unspecified whether a thread that has exited but remains unjoined counts against
 33489 {PTHREAD_THREADS_MAX}.

33490 **RETURN VALUE**

33491 If successful, the *pthread_join()* function shall return zero; otherwise, an error number shall be
 33492 returned to indicate the error.

33493 **ERRORS**33494 The *pthread_join()* function shall fail if:

33495 [EINVAL] The implementation has detected that the value specified by *thread* does not
 33496 refer to a joinable thread.

33497 [ESRCH] No thread could be found corresponding to that specified by the given thread
 33498 ID.

33499 The *pthread_join()* function may fail if:

33500 [EDEADLK] A deadlock was detected or the value of *thread* specifies the calling thread.

33501 The *pthread_join()* function shall not return an error code of [EINTR].33502 **EXAMPLES**

33503 An example of thread creation and deletion follows:

```

33504 typedef struct {
33505     int *ar;
33506     long n;
33507 } subarray;

33508 void *
33509 incer(void *arg)
33510 {
33511     long i;

33512     for (i = 0; i < ((subarray *)arg)->n; i++)
33513         ((subarray *)arg)->ar[i]++;
33514 }

33515 int main(void)
33516 {
33517     int          ar[1000000];

```



```

33518         pthread_t  th1, th2;
33519         subarray   sb1, sb2;

33520         sb1.ar = &ar[0];
33521         sb1.n  = 500000;
33522         (void) pthread_create(&th1, NULL, incer, &sb1);

33523         sb2.ar = &ar[500000];
33524         sb2.n  = 500000;
33525         (void) pthread_create(&th2, NULL, incer, &sb2);

33526         (void) pthread_join(th1, NULL);
33527         (void) pthread_join(th2, NULL);
33528         return 0;
33529     }

```

33530 APPLICATION USAGE

33531 None.

33532 RATIONALE

33533 The *pthread_join()* function is a convenience that has proven useful in multi-threaded
 33534 applications. It is true that a programmer could simulate this function if it were not provided by
 33535 passing extra state as part of the argument to the *start_routine()*. The terminating thread would
 33536 set a flag to indicate termination and broadcast a condition that is part of that state; a joining
 33537 thread would wait on that condition variable. While such a technique would allow a thread to
 33538 wait on more complex conditions (for example, waiting for multiple threads to terminate),
 33539 waiting on individual thread termination is considered widely useful. Also, including the
 33540 *pthread_join()* function in no way precludes a programmer from coding such complex waits.
 33541 Thus, while not a primitive, including *pthread_join()* in this volume of IEEE Std 1003.1-2001 was
 33542 considered valuable.

33543 The *pthread_join()* function provides a simple mechanism allowing an application to wait for a
 33544 thread to terminate. After the thread terminates, the application may then choose to clean up
 33545 resources that were used by the thread. For instance, after *pthread_join()* returns, any
 33546 application-provided stack storage could be reclaimed.

33547 The *pthread_join()* or *pthread_detach()* function should eventually be called for every thread that
 33548 is created with the *detachstate* attribute set to *PTHREAD_CREATE_JOINABLE* so that storage
 33549 associated with the thread may be reclaimed.

33550 The interaction between *pthread_join()* and cancellation is well-defined for the following
 33551 reasons:

- 33552 • The *pthread_join()* function, like all other non-async-cancel-safe functions, can only be called
 33553 with deferred cancelability type.
- 33554 • Cancellation cannot occur in the disabled cancelability state.

33555 Thus, only the default cancelability state need be considered. As specified, either the
 33556 *pthread_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the
 33557 application, since either a cancellation handler is run or *pthread_join()* returns. There are no race
 33558 conditions since *pthread_join()* was called in the deferred cancelability state.

33559 FUTURE DIRECTIONS

33560 None.

33561 **SEE ALSO**

33562 *pthread_create()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

33563 **CHANGE HISTORY**

33564 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33565 **Issue 6**

33566 The *pthread_join()* function is marked as part of the Threads option.

33567 **NAME**

33568 pthread_key_create — thread-specific data key creation

33569 **SYNOPSIS**33570 THR `#include <pthread.h>`33571 `int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));`

33572

33573 **DESCRIPTION**

33574 The *pthread_key_create()* function shall create a thread-specific data key visible to all threads in
 33575 the process. Key values provided by *pthread_key_create()* are opaque objects used to locate
 33576 thread-specific data. Although the same key value may be used by different threads, the values
 33577 bound to the key by *pthread_setspecific()* are maintained on a per-thread basis and persist for the
 33578 life of the calling thread.

33579 Upon key creation, the value NULL shall be associated with the new key in all active threads.
 33580 Upon thread creation, the value NULL shall be associated with all defined keys in the new
 33581 thread.

33582 An optional destructor function may be associated with each key value. At thread exit, if a key
 33583 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with
 33584 that key, the value of the key is set to NULL, and then the function pointed to is called with the
 33585 previously associated value as its sole argument. The order of destructor calls is unspecified if
 33586 more than one destructor exists for a thread when it exits.

33587 If, after all the destructors have been called for all non-NULL values with associated destructors,
 33588 there are still some non-NULL values with associated destructors, then the process is repeated.
 33589 If, after at least {PTHREAD_DESTRUCTOR_ITERATIONS} iterations of destructor calls for
 33590 outstanding non-NULL values, there are still some non-NULL values with associated
 33591 destructors, implementations may stop calling destructors, or they may continue calling
 33592 destructors until no non-NULL values with associated destructors exist, even though this might
 33593 result in an infinite loop.

33594 **RETURN VALUE**

33595 If successful, the *pthread_key_create()* function shall store the newly created key value at **key* and
 33596 shall return zero. Otherwise, an error number shall be returned to indicate the error.

33597 **ERRORS**33598 The *pthread_key_create()* function shall fail if:

33599 [EAGAIN] The system lacked the necessary resources to create another thread-specific
 33600 data key, or the system-imposed limit on the total number of keys per process
 33601 {PTHREAD_KEYS_MAX} has been exceeded.

33602 [ENOMEM] Insufficient memory exists to create the key.

33603 The *pthread_key_create()* function shall not return an error code of [EINTR].

33604 **EXAMPLES**

33605 The following example demonstrates a function that initializes a thread-specific data key when
 33606 it is first called, and associates a thread-specific object with each calling thread, initializing this
 33607 object when necessary.

```

33608       static pthread_key_t key;
33609       static pthread_once_t key_once = PTHREAD_ONCE_INIT;

33610       static void
33611       make_key()
33612       {
33613           (void) pthread_key_create(&key, NULL);
33614       }

33615       func()
33616       {
33617           void *ptr;

33618           (void) pthread_once(&key_once, make_key);
33619           if ((ptr = pthread_getspecific(key)) == NULL) {
33620               ptr = malloc(OBJECT_SIZE);
33621               ...
33622               (void) pthread_setspecific(key, ptr);
33623           }
33624           ...
33625       }
```

33626 Note that the key has to be initialized before *pthread_getspecific()* or *pthread_setspecific()* can be
 33627 used. The *pthread_key_create()* call could either be explicitly made in a module initialization
 33628 routine, or it can be done implicitly by the first call to a module as in this example. Any attempt
 33629 to use the key before it is initialized is a programming error, making the code below incorrect.

```

33630       static pthread_key_t key;

33631       func()
33632       {
33633           void *ptr;

33634           /* KEY NOT INITIALIZED!!! THIS WON'T WORK!!! */
33635           if ((ptr = pthread_getspecific(key)) == NULL &&
33636               pthread_setspecific(key, NULL) != 0) {
33637               pthread_key_create(&key, NULL);
33638               ...
33639           }
33640       }
```

33641 **APPLICATION USAGE**

33642 None.

33643 **RATIONALE**33644 **Destructor Functions**

33645 Normally, the value bound to a key on behalf of a particular thread is a pointer to storage
 33646 allocated dynamically on behalf of the calling thread. The destructor functions specified with
 33647 *pthread_key_create()* are intended to be used to free this storage when the thread exits. Thread
 33648 cancellation cleanup handlers cannot be used for this purpose because thread-specific data may
 33649 persist outside the lexical scope in which the cancellation cleanup handlers operate.

33650 If the value associated with a key needs to be updated during the lifetime of the thread, it may
 33651 be necessary to release the storage associated with the old value before the new value is bound.
 33652 Although the *pthread_setspecific()* function could do this automatically, this feature is not needed
 33653 often enough to justify the added complexity. Instead, the programmer is responsible for freeing
 33654 the stale storage:

```
33655 pthread_getspecific(key, &old);
33656 new = allocate();
33657 destructor(old);
33658 pthread_setspecific(key, new);
```

33659 **Note:** The above example could leak storage if run with asynchronous cancellation enabled. No such
 33660 problems occur in the default cancellation state if no cancellation points occur between the get
 33661 and set.

33662 There is no notion of a destructor-safe function. If an application does not call *pthread_exit()*
 33663 from a signal handler, or if it blocks any signal whose handler may call *pthread_exit()* while
 33664 calling async-unsafe functions, all functions may be safely called from destructors.

33665 **Non-Idempotent Data Key Creation**

33666 There were requests to make *pthread_key_create()* idempotent with respect to a given *key* address
 33667 parameter. This would allow applications to call *pthread_key_create()* multiple times for a given
 33668 *key* address and be guaranteed that only one key would be created. Doing so would require the
 33669 key value to be previously initialized (possibly at compile time) to a known null value and
 33670 would require that implicit mutual-exclusion be performed based on the address and contents of
 33671 the *key* parameter in order to guarantee that exactly one key would be created.

33672 Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread_key_create()*.
 33673 On many implementations, implicit mutual-exclusion would also have to be performed by
 33674 *pthread_getspecific()* and *pthread_setspecific()* in order to guard against using incompletely stored
 33675 or not-yet-visible key values. This could significantly increase the cost of important operations,
 33676 particularly *pthread_getspecific()*.

33677 Thus, this proposal was rejected. The *pthread_key_create()* function performs no implicit
 33678 synchronization. It is the responsibility of the programmer to ensure that it is called exactly once
 33679 per key before use of the key. Several straightforward mechanisms can already be used to
 33680 accomplish this, including calling explicit module initialization functions, using mutexes, and
 33681 using *pthread_once()*. This places no significant burden on the programmer, introduces no
 33682 possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows
 33683 commonly used thread-specific data operations to be more efficient.

33684 **FUTURE DIRECTIONS**

33685 None.

33686 SEE ALSO

33687 *pthread_getspecific()*, *pthread_key_delete()*, the Base Definitions volume of IEEE Std 1003.1-2001,
33688 **<pthread.h>**

33689 CHANGE HISTORY

33690 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33691 Issue 6

33692 The *pthread_key_create()* function is marked as part of the Threads option.

33693 IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

33694 **NAME**

33695 pthread_key_delete — thread-specific data key deletion

33696 **SYNOPSIS**

33697 THR #include <pthread.h>

33698 int pthread_key_delete(pthread_key_t key);

33699

33700 **DESCRIPTION**

33701 The *pthread_key_delete()* function shall delete a thread-specific data key previously returned by
 33702 *pthread_key_create()*. The thread-specific data values associated with *key* need not be NULL at
 33703 the time *pthread_key_delete()* is called. It is the responsibility of the application to free any
 33704 application storage or perform any cleanup actions for data structures related to the deleted key
 33705 or associated thread-specific data in any threads; this cleanup can be done either before or after
 33706 *pthread_key_delete()* is called. Any attempt to use *key* following the call to *pthread_key_delete()*
 33707 results in undefined behavior.

33708 The *pthread_key_delete()* function shall be callable from within destructor functions. No
 33709 destructor functions shall be invoked by *pthread_key_delete()*. Any destructor function that may
 33710 have been associated with *key* shall no longer be called upon thread exit.

33711 **RETURN VALUE**

33712 If successful, the *pthread_key_delete()* function shall return zero; otherwise, an error number shall
 33713 be returned to indicate the error.

33714 **ERRORS**33715 The *pthread_key_delete()* function may fail if:33716 [EINVAL] The *key* value is invalid.33717 The *pthread_key_delete()* function shall not return an error code of [EINTR].33718 **EXAMPLES**

33719 None.

33720 **APPLICATION USAGE**

33721 None.

33722 **RATIONALE**

33723 A thread-specific data key deletion function has been included in order to allow the resources
 33724 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys
 33725 can arise, among other scenarios, when a dynamically loaded module that allocated a key is
 33726 unloaded.

33727 Conforming applications are responsible for performing any cleanup actions needed for data
 33728 structures associated with the key to be deleted, including data referenced by thread-specific
 33729 data values. No such cleanup is done by *pthread_key_delete()*. In particular, destructor functions
 33730 are not called. There are several reasons for this division of responsibility:

- 33731 1. The associated destructor functions used to free thread-specific data at thread exit time are
 33732 only guaranteed to work correctly when called in the thread that allocated the thread-
 33733 specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot
 33734 be used to free thread-specific data in other threads at key deletion time. Attempting to
 33735 have them called by other threads at key deletion time would require other threads to be
 33736 asynchronously interrupted. But since interrupted threads could be in an arbitrary state,
 33737 including holding locks necessary for the destructor to run, this approach would fail. In
 33738 general, there is no safe mechanism whereby an implementation could free thread-specific
 33739 data at key deletion time.

33740 2. Even if there were a means of safely freeing thread-specific data associated with keys to be
33741 deleted, doing so would require that implementations be able to enumerate the threads
33742 with non-NULL data and potentially keep them from creating more thread-specific data
33743 while the key deletion is occurring. This special case could cause extra synchronization in
33744 the normal case, which would otherwise be unnecessary.

33745 For an application to know that it is safe to delete a key, it has to know that all the threads that
33746 might potentially ever use the key do not attempt to use it again. For example, it could know this
33747 if all the client threads have called a cleanup procedure declaring that they are through with the
33748 module that is being shut down, perhaps by setting a reference count to zero.

33749 **FUTURE DIRECTIONS**

33750 None.

33751 **SEE ALSO**

33752 *pthread_key_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

33753 **CHANGE HISTORY**

33754 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33755 **Issue 6**

33756 The *pthread_key_delete()* function is marked as part of the Threads option.

33757 **NAME**

33758 pthread_kill — send a signal to a thread

33759 **SYNOPSIS**

33760 THR #include <signal.h>

33761 int pthread_kill(pthread_t thread, int sig);

33762

33763 **DESCRIPTION**33764 The *pthread_kill()* function shall request that a signal be delivered to the specified thread.33765 As in *kill()*, if *sig* is zero, error checking shall be performed but no signal shall actually be sent.33766 **RETURN VALUE**33767 Upon successful completion, the function shall return a value of zero. Otherwise, the function
33768 shall return an error number. If the *pthread_kill()* function fails, no signal shall be sent.33769 **ERRORS**33770 The *pthread_kill()* function shall fail if:33771 [ESRCH] No thread could be found corresponding to that specified by the given thread
33772 ID.33773 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.33774 The *pthread_kill()* function shall not return an error code of [EINTR].33775 **EXAMPLES**

33776 None.

33777 **APPLICATION USAGE**33778 The *pthread_kill()* function provides a mechanism for asynchronously directing a signal at a
33779 thread in the calling process. This could be used, for example, by one thread to affect broadcast
33780 delivery of a signal to a set of threads.33781 Note that *pthread_kill()* only causes the signal to be handled in the context of the given thread;
33782 the signal action (termination or stopping) affects the process as a whole.33783 **RATIONALE**

33784 None.

33785 **FUTURE DIRECTIONS**

33786 None.

33787 **SEE ALSO**33788 *kill()*, *pthread_self()*, *raise()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>33789 **CHANGE HISTORY**

33790 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33791 **Issue 6**33792 The *pthread_kill()* function is marked as part of the Threads option.

33793 The APPLICATION USAGE section is added.

33794 **NAME**

33795 pthread_mutex_destroy, pthread_mutex_init — destroy and initialize a mutex

33796 **SYNOPSIS**

33797 THR #include <pthread.h>

```

33798 int pthread_mutex_destroy(pthread_mutex_t *mutex);
33799 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
33800     const pthread_mutexattr_t *restrict attr);
33801 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
33802

```

33803 **DESCRIPTION**

33804 The *pthread_mutex_destroy()* function shall destroy the mutex object referenced by *mutex*; the
 33805 mutex object becomes, in effect, uninitialized. An implementation may cause
 33806 *pthread_mutex_destroy()* to set the object referenced by *mutex* to an invalid value. A destroyed
 33807 mutex object can be reinitialized using *pthread_mutex_init()*; the results of otherwise referencing
 33808 the object after it has been destroyed are undefined.

33809 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked
 33810 mutex results in undefined behavior.

33811 The *pthread_mutex_init()* function shall initialize the mutex referenced by *mutex* with attributes
 33812 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the
 33813 same as passing the address of a default mutex attributes object. Upon successful initialization,
 33814 the state of the mutex becomes initialized and unlocked.

33815 Only *mutex* itself may be used for performing synchronization. The result of referring to copies
 33816 of *mutex* in calls to *pthread_mutex_lock()*, *pthread_mutex_trylock()*, *pthread_mutex_unlock()*, and
 33817 *pthread_mutex_destroy()* is undefined.

33818 Attempting to initialize an already initialized mutex results in undefined behavior.

33819 In cases where default mutex attributes are appropriate, the macro
 33820 PTHREAD_MUTEX_INITIALIZER can be used to initialize mutexes that are statically allocated.
 33821 The effect shall be equivalent to dynamic initialization by a call to *pthread_mutex_init()* with
 33822 parameter *attr* specified as NULL, except that no error checks are performed.

33823 **RETURN VALUE**

33824 If successful, the *pthread_mutex_destroy()* and *pthread_mutex_init()* functions shall return zero;
 33825 otherwise, an error number shall be returned to indicate the error.

33826 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed
 33827 immediately at the beginning of processing for the function and shall cause an error return prior
 33828 to modifying the state of the mutex specified by *mutex*.

33829 **ERRORS**

33830 The *pthread_mutex_destroy()* function may fail if:

33831 [EBUSY] The implementation has detected an attempt to destroy the object referenced
 33832 by *mutex* while it is locked or referenced (for example, while being used in a
 33833 *pthread_cond_timedwait()* or *pthread_cond_wait()*) by another thread.

33834 [EINVAL] The value specified by *mutex* is invalid.

33835 The *pthread_mutex_init()* function shall fail if:

33836 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
 33837 another mutex.

- 33838 [ENOMEM] Insufficient memory exists to initialize the mutex.
- 33839 [EPERM] The caller does not have the privilege to perform the operation.
- 33840 The *pthread_mutex_init()* function may fail if:
- 33841 [EBUSY] The implementation has detected an attempt to reinitialize the object
33842 referenced by *mutex*, a previously initialized, but not yet destroyed, mutex.
- 33843 [EINVAL] The value specified by *attr* is invalid.
- 33844 These functions shall not return an error code of [EINTR].

33845 EXAMPLES

33846 None.

33847 APPLICATION USAGE

33848 None.

33849 RATIONALE**33850 Alternate Implementations Possible**

33851 This volume of IEEE Std 1003.1-2001 supports several alternative implementations of mutexes.
33852 An implementation may store the lock directly in the object of type **pthread_mutex_t**.
33853 Alternatively, an implementation may store the lock in the heap and merely store a pointer,
33854 handle, or unique ID in the mutex object. Either implementation has advantages or may be
33855 required on certain hardware configurations. So that portable code can be written that is
33856 invariant to this choice, this volume of IEEE Std 1003.1-2001 does not define assignment or
33857 equality for this type, and it uses the term “initialize” to reinforce the (more restrictive) notion
33858 that the lock may actually reside in the mutex object itself.

33859 Note that this precludes an over-specification of the type of the mutex or condition variable and
33860 motivates the opaqueness of the type.

33861 An implementation is permitted, but not required, to have *pthread_mutex_destroy()* store an
33862 illegal value into the mutex. This may help detect erroneous programs that try to lock (or
33863 otherwise reference) a mutex that has already been destroyed.

33864 Tradeoff Between Error Checks and Performance Supported

33865 Many of the error checks were made optional in order to let implementations trade off
33866 performance *versus* degree of error checking according to the needs of their specific applications
33867 and execution environment. As a general rule, errors or conditions caused by the system (such as
33868 insufficient memory) always need to be reported, but errors due to an erroneously coded
33869 application (such as failing to provide adequate synchronization to prevent a mutex from being
33870 deleted while in use) are made optional.

33871 A wide range of implementations is thus made possible. For example, an implementation
33872 intended for application debugging may implement all of the error checks, but an
33873 implementation running a single, provably correct application under very tight performance
33874 constraints in an embedded computer might implement minimal checks. An implementation
33875 might even be provided in two versions, similar to the options that compilers provide: a full-
33876 checking, but slower version; and a limited-checking, but faster version. To forbid this
33877 optionality would be a disservice to users.

33878 By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly
33879 coded) application might do, and by defining that resource-not-available errors are mandatory,
33880 this volume of IEEE Std 1003.1-2001 ensures that a fully-conforming application is portable

across the full range of implementations, while not forcing all implementations to add overhead to check for numerous things that a correct program never does.

Why No Limits are Defined

Defining symbols for the maximum number of mutexes and condition variables was considered but rejected because the number of these objects may change dynamically. Furthermore, many implementations place these objects into application memory; thus, there is no explicit maximum.

Static Initializers for Mutexes and Condition Variables

Providing for static initialization of statically allocated synchronization objects allows modules with private static synchronization variables to avoid runtime initialization tests and overhead. Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in C libraries, where for various reasons the design calls for self-initialization instead of requiring an explicit module initialization function to be called. An example use of static initialization follows.

Without static initialization, a self-initializing routine *foo()* might look as follows:

```
static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
static pthread_mutex_t foo_mutex;

void foo_init()
{
    pthread_mutex_init(&foo_mutex, NULL);
}

void foo()
{
    pthread_once(&foo_once, foo_init);
    pthread_mutex_lock(&foo_mutex);
    /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

With static initialization, the same routine could be coded as follows:

```
static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;

void foo()
{
    pthread_mutex_lock(&foo_mutex);
    /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

Note that the static initialization both eliminates the need for the initialization test inside *pthread_once()* and the fetch of *&foo_mutex* to learn the address to be passed to *pthread_mutex_lock()* or *pthread_mutex_unlock()*.

Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a large class of systems; those where the (entire) synchronization object can be stored in application memory.

Yet the locking performance question is likely to be raised for machines that require mutexes to be allocated out of special memory. Such machines actually have to have mutexes and possibly

condition variables contain pointers to the actual hardware locks. For static initialization to work on such machines, *pthread_mutex_lock()* also has to test whether or not the pointer to the actual lock has been allocated. If it has not, *pthread_mutex_lock()* has to initialize it before use. The reservation of such resources can be made when the program is loaded, and hence return codes have not been added to mutex locking and condition variable waiting to indicate failure to complete initialization.

This runtime test in *pthread_mutex_lock()* would at first seem to be extra work; an extra test is required to see whether the pointer has been initialized. On most machines this would actually be implemented as a fetch of the pointer, testing the pointer against zero, and then using the pointer if it has already been initialized. While the test might seem to add extra work, the extra effort of testing a register is usually negligible since no extra memory references are actually done. As more and more machines provide caches, the real expenses are memory references, not instructions executed.

Alternatively, depending on the machine architecture, there are often ways to eliminate *all* overhead in the most important case: on the lock operations that occur *after* the lock has been initialized. This can be done by shifting more overhead to the less frequent operation: initialization. Since out-of-line mutex allocation also means that an address has to be dereferenced to find the actual lock, one technique that is widely applicable is to have static initialization store a bogus value for that address; in particular, an address that causes a machine fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity checks can be done, and then the correct address for the actual lock can be filled in. Subsequent lock operations incur no extra overhead since they do not “fault”. This is merely one technique that can be used to support static initialization, while not adversely affecting the performance of lock acquisition. No doubt there are other techniques that are highly machine-dependent.

The locking overhead for machines doing out-of-line mutex allocation is thus similar for modules being implicitly initialized, where it is improved for those doing mutex allocation entirely inline. The inline case is thus made much faster, and the out-of-line case is not significantly worse.

Besides the issue of locking performance for such machines, a concern is raised that it is possible that threads would serialize contending for initialization locks when attempting to finish initializing statically allocated mutexes. (Such finishing would typically involve taking an internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing the internal lock.) First, many implementations would reduce such serialization by hashing on the mutex address. Second, such serialization can only occur a bounded number of times. In particular, it can happen at most as many times as there are statically allocated synchronization objects. Dynamically allocated objects would still be initialized via *pthread_mutex_init()* or *pthread_cond_init()*.

Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient performance for an application on some implementation, the application can avoid static initialization altogether by explicitly initializing all synchronization objects with the corresponding *pthread_*_init()* functions, which are supported by all implementations. An implementation can also document the tradeoffs and advise which initialization technique is more efficient for that particular implementation.

Destroying Mutexes

A mutex can be destroyed immediately after it is unlocked. For example, consider the following code:

```
struct obj {
    pthread_mutex_t om;
    int refcnt;
    ...
};

obj_done(struct obj *op)
{
    pthread_mutex_lock(&op->om);
    if (--op->refcnt == 0) {
        pthread_mutex_unlock(&op->om);
        (A) pthread_mutex_destroy(&op->om);
        (B) free(op);
    } else
        (C) pthread_mutex_unlock(&op->om);
}
```

In this case *obj* is reference counted and *obj_done()* is called whenever a reference to the object is dropped. Implementations are required to allow an object to be destroyed and freed and potentially unmapped (for example, lines A and B) immediately after the object is unlocked (line C).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_getprioceiling(), *pthread_mutex_lock()*, *pthread_mutex_timedlock()*, *pthread_mutexattr_getpshared()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_mutex_destroy()* and *pthread_mutex_init()* functions are marked as part of the Threads option.

The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

The **restrict** keyword is added to the *pthread_mutex_init()* prototype for alignment with the ISO/IEC 9899:1999 standard.

34006 NAME

34007 pthread_mutex_getprioceiling, pthread_mutex_setprioceiling — get and set the priority ceiling
 34008 of a mutex (**REALTIME THREADS**)

34009 SYNOPSIS

34010 THR TPP #include <pthread.h>

```
34011 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
34012 int *restrict prioceiling);
34013 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
34014 int prioceiling, int *restrict old_ceiling);
34015
```

34016 DESCRIPTION

34017 The *pthread_mutex_getprioceiling()* function shall return the current priority ceiling of the mutex.

34018 The *pthread_mutex_setprioceiling()* function shall either lock the mutex if it is unlocked, or block
 34019 until it can successfully lock the mutex, then it shall change the mutex's priority ceiling and
 34020 release the mutex. When the change is successful, the previous value of the priority ceiling shall
 34021 be returned in *old_ceiling*. The process of locking the mutex need not adhere to the priority
 34022 protect protocol.

34023 If the *pthread_mutex_setprioceiling()* function fails, the mutex priority ceiling shall not be
 34024 changed.

34025 RETURN VALUE

34026 If successful, the *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions shall
 34027 return zero; otherwise, an error number shall be returned to indicate the error.

34028 ERRORS

34029 The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions may fail if:

- 34030 [EINVAL] The priority requested by *prioceiling* is out of range.
- 34031 [EINVAL] The value specified by *mutex* does not refer to a currently existing mutex.
- 34032 [EPERM] The caller does not have the privilege to perform the operation.
- 34033 These functions shall not return an error code of [EINTR].

34034 EXAMPLES

34035 None.

34036 APPLICATION USAGE

34037 None.

34038 RATIONALE

34039 None.

34040 FUTURE DIRECTIONS

34041 None.

34042 SEE ALSO

34043 *pthread_mutex_destroy()*, *pthread_mutex_lock()*, *pthread_mutex_timedlock()*, the Base Definitions
 34044 volume of IEEE Std 1003.1-2001, <**pthread.h**>

34045 CHANGE HISTORY

- 34046 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 34047 Marked as part of the Realtime Threads Feature Group.

Issue 6

The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions are marked as part of the Threads and Thread Priority Protection options.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection option.

The [ENOSYS] error denoting non-support of the priority ceiling protocol for mutexes has been removed. This is because if the implementation provides the functions (regardless of whether `_POSIX_PTHREAD_PRIO_PROTECT` is defined), they must function as in the DESCRIPTION and therefore the priority ceiling protocol for mutexes is supported.

The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* prototypes for alignment with the ISO / IEC 9899:1999 standard.

34061 **NAME**

34062 pthread_mutex_init — initialize a mutex

34063 **SYNOPSIS**

34064 THR #include <pthread.h>

34065 int pthread_mutex_init(pthread_mutex_t *restrict mutex,

34066 const pthread_mutexattr_t *restrict attr);

34067 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

34068

34069 **DESCRIPTION**34070 Refer to *pthread_mutex_destroy()*.

34071 **NAME**

34072 pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a
 34073 mutex

34074 **SYNOPSIS**

34075 THR #include <pthread.h>

```
34076 int pthread_mutex_lock(pthread_mutex_t *mutex);
34077 int pthread_mutex_trylock(pthread_mutex_t *mutex);
34078 int pthread_mutex_unlock(pthread_mutex_t *mutex);
34079
```

34080 **DESCRIPTION**

34081 The mutex object referenced by *mutex* shall be locked by calling *pthread_mutex_lock()*. If the
 34082 mutex is already locked, the calling thread shall block until the mutex becomes available. This
 34083 operation shall return with the mutex object referenced by *mutex* in the locked state with the
 34084 calling thread as its owner.

34085 XSI If the mutex type is PTHREAD_MUTEX_NORMAL, deadlock detection shall not be provided.
 34086 Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it
 34087 has not locked or a mutex which is unlocked, undefined behavior results.

34088 If the mutex type is PTHREAD_MUTEX_ERRORCHECK, then error checking shall be provided.
 34089 If a thread attempts to relock a mutex that it has already locked, an error shall be returned. If a
 34090 thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error
 34091 shall be returned.

34092 If the mutex type is PTHREAD_MUTEX_RECURSIVE, then the mutex shall maintain the
 34093 concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock
 34094 count shall be set to one. Every time a thread relocks this mutex, the lock count shall be
 34095 incremented by one. Each time the thread unlocks the mutex, the lock count shall be
 34096 decremented by one. When the lock count reaches zero, the mutex shall become available for
 34097 other threads to acquire. If a thread attempts to unlock a mutex that it has not locked or a mutex
 34098 which is unlocked, an error shall be returned.

34099 If the mutex type is PTHREAD_MUTEX_DEFAULT, attempting to recursively lock the mutex
 34100 results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling
 34101 thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in
 34102 undefined behavior.

34103 The *pthread_mutex_trylock()* function shall be equivalent to *pthread_mutex_lock()*, except that if
 34104 the mutex object referenced by *mutex* is currently locked (by any thread, including the current
 34105 thread), the call shall return immediately. If the mutex type is PTHREAD_MUTEX_RECURSIVE
 34106 and the mutex is currently owned by the calling thread, the mutex lock count shall be
 34107 incremented by one and the *pthread_mutex_trylock()* function shall immediately return success.

34108 XSI The *pthread_mutex_unlock()* function shall release the mutex object referenced by *mutex*. The
 34109 manner in which a mutex is released is dependent upon the mutex's type attribute. If there are
 34110 threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock()* is called,
 34111 resulting in the mutex becoming available, the scheduling policy shall determine which thread
 34112 shall acquire the mutex.

34113 XSI (In the case of PTHREAD_MUTEX_RECURSIVE mutexes, the mutex shall become available
 34114 when the count reaches zero and the calling thread no longer has any locks on this mutex.)

34115 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the
 34116 thread shall resume waiting for the mutex as if it was not interrupted.

34117 RETURN VALUE

34118 If successful, the *pthread_mutex_lock()* and *pthread_mutex_unlock()* functions shall return zero;
 34119 otherwise, an error number shall be returned to indicate the error.

34120 The *pthread_mutex_trylock()* function shall return zero if a lock on the mutex object referenced by
 34121 *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

34122 ERRORS

34123 The *pthread_mutex_lock()* and *pthread_mutex_trylock()* functions shall fail if:

34124 [EINVAL] The *mutex* was created with the protocol attribute having the value
 34125 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
 34126 the mutex's current priority ceiling.

34127 The *pthread_mutex_trylock()* function shall fail if:

34128 [EBUSY] The *mutex* could not be acquired because it was already locked.

34129 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions may
 34130 fail if:

34131 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34132 xsi [EAGAIN] The mutex could not be acquired because the maximum number of recursive
 34133 locks for *mutex* has been exceeded.

34134 The *pthread_mutex_lock()* function may fail if:

34135 [EDEADLK] The current thread already owns the mutex.

34136 The *pthread_mutex_unlock()* function may fail if:

34137 [EPERM] The current thread does not own the mutex.

34138 These functions shall not return an error code of [EINTR].

34139 EXAMPLES

34140 None.

34141 APPLICATION USAGE

34142 None.

34143 RATIONALE

34144 Mutex objects are intended to serve as a low-level primitive from which other thread
 34145 synchronization functions can be built. As such, the implementation of mutexes should be as
 34146 efficient as possible, and this has ramifications on the features available at the interface.

34147 The mutex functions and the particular default settings of the mutex attributes have been
 34148 motivated by the desire to not preclude fast, inlined implementations of mutex locking and
 34149 unlocking.

34150 For example, deadlocking on a double-lock is explicitly allowed behavior in order to avoid
 34151 requiring more overhead in the basic mechanism than is absolutely necessary. (More “friendly”
 34152 mutexes that detect deadlock or that allow multiple locking by the same thread are easily
 34153 constructed by the user via the other mechanisms provided. For example, *pthread_self()* can be
 34154 used to record mutex ownership.) Implementations might also choose to provide such extended
 34155 features as options via special mutex attributes.

34156 Since most attributes only need to be checked when a thread is going to be blocked, the use of
 34157 attributes does not slow the (common) mutex-locking case.

34158 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it
34159 would require storing the current thread ID when each mutex is locked, and this could incur
34160 unacceptable levels of overhead. Similar arguments apply to a *mutex_tryunlock* operation.

34161 FUTURE DIRECTIONS

34162 None.

34163 SEE ALSO

34164 *pthread_mutex_destroy()*, *pthread_mutex_timedlock()*, the Base Definitions volume of
34165 IEEE Std 1003.1-2001, <pthread.h>

34166 CHANGE HISTORY

34167 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34168 Issue 6

34169 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions are
34170 marked as part of the Threads option.

34171 The following new requirements on POSIX implementations derive from alignment with the
34172 Single UNIX Specification:

- 34173 • The behavior when attempting to relock a mutex is defined.

34174 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with
34175 IEEE Std 1003.1d-1999.

34176 **NAME**

34177 pthread_mutex_setprioceiling — change the priority ceiling of a mutex (**REALTIME**
34178 **THREADS**)

34179 **SYNOPSIS**

34180 THR TPP #include <pthread.h>

34181 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
34182 int prioceiling, int *restrict old_ceiling);

34183

34184 **DESCRIPTION**

34185 Refer to *pthread_mutex_getprioceiling()*.

34186 **NAME**34187 pthread_mutex_timedlock — lock a mutex (**ADVANCED REALTIME**)34188 **SYNOPSIS**

34189 THR TMO #include <pthread.h>

34190 #include <time.h>

```
34191 int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
34192     const struct timespec *restrict abs_timeout);
```

34193

34194 **DESCRIPTION**

34195 The *pthread_mutex_timedlock()* function shall lock the mutex object referenced by *mutex*. If the
 34196 mutex is already locked, the calling thread shall block until the mutex becomes available as in
 34197 the *pthread_mutex_lock()* function. If the mutex cannot be locked without waiting for another
 34198 thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

34199 The timeout shall expire when the absolute time specified by *abs_timeout* passes, as measured by
 34200 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 34201 *abs_timeout*), or if the absolute time specified by *abs_timeout* has already been passed at the time
 34202 of the call.

34203 TMR If the Timers option is supported, the timeout shall be based on the CLOCK_REALTIME clock; if
 34204 the Timers option is not supported, the timeout shall be based on the system clock as returned
 34205 by the *time()* function.

34206 The resolution of the timeout shall be the resolution of the clock on which it is based. The
 34207 **timespec** data type is defined in the <time.h> header.

34208 Under no circumstance shall the function fail with a timeout if the mutex can be locked
 34209 immediately. The validity of the *abs_timeout* parameter need not be checked if the mutex can be
 34210 locked immediately.

34211 As a consequence of the priority inheritance rules (for mutexes initialized with the
 34212 PRIO_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the
 34213 priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this
 34214 thread is no longer among the threads waiting for the mutex.

34215 **RETURN VALUE**

34216 If successful, the *pthread_mutex_timedlock()* function shall return zero; otherwise, an error
 34217 number shall be returned to indicate the error.

34218 **ERRORS**

34219 The *pthread_mutex_timedlock()* function shall fail if:

34220 [EINVAL] The mutex was created with the protocol attribute having the value
 34221 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
 34222 the mutex' current priority ceiling.

34223 [EINVAL] The process or thread would have blocked, and the *abs_timeout* parameter
 34224 specified a nanoseconds field value less than zero or greater than or equal to
 34225 1 000 million.

34226 [ETIMEDOUT] The mutex could not be locked before the specified timeout expired.

34227 The *pthread_mutex_timedlock()* function may fail if:

34228 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34229 xSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive
34230 locks for *mutex* has been exceeded.

34231 [EDEADLK] The current thread already owns the mutex.

34232 This function shall not return an error code of [EINTR].

34233 **EXAMPLES**

34234 None.

34235 **APPLICATION USAGE**

34236 The *pthread_mutex_timedlock()* function is part of the Threads and Timeouts options and need
34237 not be provided on all implementations.

34238 **RATIONALE**

34239 None.

34240 **FUTURE DIRECTIONS**

34241 None.

34242 **SEE ALSO**

34243 *pthread_mutex_destroy()*, *pthread_mutex_lock()*, *pthread_mutex_trylock()*, *time()*, the Base
34244 Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>, <**time.h**>

34245 **CHANGE HISTORY**

34246 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

34247 NAME

34248 pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

34249 SYNOPSIS

34250 THR `#include <pthread.h>`

34251 `int pthread_mutex_trylock(pthread_mutex_t *mutex);`

34252 `int pthread_mutex_unlock(pthread_mutex_t *mutex);`

34253

34254 DESCRIPTION

34255 Refer to *pthread_mutex_lock()*.

34256 NAME

34257 pthread_mutexattr_destroy, pthread_mutexattr_init — destroy and initialize the mutex
 34258 attributes object

34259 SYNOPSIS

34260 THR #include <pthread.h>

34261 int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);

34262 int pthread_mutexattr_init(pthread_mutexattr_t *attr);

34263

34264 DESCRIPTION

34265 The *pthread_mutexattr_destroy()* function shall destroy a mutex attributes object; the object
 34266 becomes, in effect, uninitialized. An implementation may cause *pthread_mutexattr_destroy()* to
 34267 set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be
 34268 reinitialized using *pthread_mutexattr_init()*; the results of otherwise referencing the object after it
 34269 has been destroyed are undefined.

34270 The *pthread_mutexattr_init()* function shall initialize a mutex attributes object *attr* with the
 34271 default value for all of the attributes defined by the implementation.

34272 Results are undefined if *pthread_mutexattr_init()* is called specifying an already initialized *attr*
 34273 attributes object.

34274 After a mutex attributes object has been used to initialize one or more mutexes, any function
 34275 affecting the attributes object (including destruction) shall not affect any previously initialized
 34276 mutexes.

34277 RETURN VALUE

34278 Upon successful completion, *pthread_mutexattr_destroy()* and *pthread_mutexattr_init()* shall
 34279 return zero; otherwise, an error number shall be returned to indicate the error.

34280 ERRORS

34281 The *pthread_mutexattr_destroy()* function may fail if:

34282 [EINVAL] The value specified by *attr* is invalid.

34283 The *pthread_mutexattr_init()* function shall fail if:

34284 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

34285 These functions shall not return an error code of [EINTR].

34286 EXAMPLES

34287 None.

34288 APPLICATION USAGE

34289 None.

34290 RATIONALE

34291 See *pthread_attr_init()* for a general explanation of attributes. Attributes objects allow
 34292 implementations to experiment with useful extensions and permit extension of this volume of
 34293 IEEE Std 1003.1-2001 without changing the existing functions. Thus, they provide for future
 34294 extensibility of this volume of IEEE Std 1003.1-2001 and reduce the temptation to standardize
 34295 prematurely on semantics that are not yet widely implemented or understood.

34296 Examples of possible additional mutex attributes that have been discussed are *spin_only*,
 34297 *limited_spin*, *no_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:
 34298 recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes
 34299 would transparently keep records of queue length, wait time, and so on.) Since there is not yet

wide agreement on the usefulness of these resulting from shared implementation and usage experience, they are not yet specified in this volume of IEEE Std 1003.1-2001. Mutex attributes objects, however, make it possible to test out these concepts for possible standardization at a later time.

Mutex Attributes and Performance

Care has been taken to ensure that the default values of the mutex attributes have been defined such that mutexes initialized with the defaults have simple enough semantics so that the locking and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few other basic instructions).

There is at least one implementation method that can be used to reduce the cost of testing at lock-time if a mutex has non-default attributes. One such method that an implementation can employ (and this can be made fully transparent to fully conforming POSIX applications) is to secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to lock such a mutex causes the implementation to branch to the “slow path” as if the mutex were unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-default mutex. The underlying unlock operation is more complicated since the implementation never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending on the hardware, there may be certain optimizations that can be used so that whatever mutex attributes are considered “most frequently used” can be processed most efficiently.

Process Shared Memory and Synchronization

The existence of memory mapping functions in this volume of IEEE Std 1003.1-2001 leads to the possibility that an application may allocate the synchronization objects from this section in memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

In order to permit such usage, while at the same time keeping the usual case (that is, usage within a single process) efficient, a *process-shared* option has been defined.

If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the *process-shared* attribute can be used to indicate that mutexes or condition variables may be accessed by threads of multiple processes.

The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared* attribute so that the most efficient forms of these synchronization objects are created by default.

Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-shared* attribute may only be operated on by threads in the process that initialized them. Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-shared* attribute may be operated on by any thread in any process that has access to it. In particular, these processes may exist beyond the lifetime of the initializing process. For example, the following code implements a simple counting semaphore in a mapped file that may be used by many processes.

```
/* sem.h */
struct semaphore {
    pthread_mutex_t lock;
    pthread_cond_t nonzero;
    unsigned count;
};
typedef struct semaphore semaphore_t;

semaphore_t *semaphore_create(char *semaphore_name);
semaphore_t *semaphore_open(char *semaphore_name);
```

```

34346 void semaphore_post(semaphore_t *semap);
34347 void semaphore_wait(semaphore_t *semap);
34348 void semaphore_close(semaphore_t *semap);

34349 /* sem.c */
34350 #include <sys/types.h>
34351 #include <sys/stat.h>
34352 #include <sys/mman.h>
34353 #include <fcntl.h>
34354 #include <pthread.h>
34355 #include "sem.h"

34356 semaphore_t *
34357 semaphore_create(char *semaphore_name)
34358 {
34359     int fd;
34360     semaphore_t *semap;
34361     pthread_mutexattr_t psharedm;
34362     pthread_condattr_t psharedc;

34363     fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
34364     if (fd < 0)
34365         return (NULL);
34366     (void) ftruncate(fd, sizeof(semaphore_t));
34367     (void) pthread_mutexattr_init(&psharedm);
34368     (void) pthread_mutexattr_setpshared(&psharedm,
34369         PTHREAD_PROCESS_SHARED);
34370     (void) pthread_condattr_init(&psharedc);
34371     (void) pthread_condattr_setpshared(&psharedc,
34372         PTHREAD_PROCESS_SHARED);
34373     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34374         PROT_READ | PROT_WRITE, MAP_SHARED,
34375         fd, 0);
34376     close (fd);
34377     (void) pthread_mutex_init(&semap->lock, &psharedm);
34378     (void) pthread_cond_init(&semap->nonzero, &psharedc);
34379     semap->count = 0;
34380     return (semap);
34381 }

34382 semaphore_t *
34383 semaphore_open(char *semaphore_name)
34384 {
34385     int fd;
34386     semaphore_t *semap;

34387     fd = open(semaphore_name, O_RDWR, 0666);
34388     if (fd < 0)
34389         return (NULL);
34390     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34391         PROT_READ | PROT_WRITE, MAP_SHARED,
34392         fd, 0);
34393     close (fd);
34394     return (semap);
34395 }

```

```

34396 void
34397 semaphore_post(semaphore_t *semap)
34398 {
34399     pthread_mutex_lock(&semap->lock);
34400     if (semap->count == 0)
34401         pthread_cond_signal(&semap->nonzero);
34402     semap->count++;
34403     pthread_mutex_unlock(&semap->lock);
34404 }
34405 void
34406 semaphore_wait(semaphore_t *semap)
34407 {
34408     pthread_mutex_lock(&semap->lock);
34409     while (semap->count == 0)
34410         pthread_cond_wait(&semap->nonzero, &semap->lock);
34411     semap->count--;
34412     pthread_mutex_unlock(&semap->lock);
34413 }
34414 void
34415 semaphore_close(semaphore_t *semap)
34416 {
34417     munmap((void *) semap, sizeof(semaphore_t));
34418 }

```

The following code is for three separate processes that create, post, and wait on a semaphore in the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and decrement the counting semaphore (waiting and waking as required) even though they did not initialize the semaphore.

```

34423 /* create.c */
34424 #include "pthread.h"
34425 #include "sem.h"
34426 int
34427 main()
34428 {
34429     semaphore_t *semap;
34430     semap = semaphore_create("/tmp/semaphore");
34431     if (semap == NULL)
34432         exit(1);
34433     semaphore_close(semap);
34434     return (0);
34435 }
34436 /* post */
34437 #include "pthread.h"
34438 #include "sem.h"
34439 int
34440 main()
34441 {
34442     semaphore_t *semap;

```

```

34443         semap = semaphore_open("/tmp/semaphore");
34444         if (semap == NULL)
34445             exit(1);
34446         semaphore_post(semap);
34447         semaphore_close(semap);
34448         return (0);
34449     }

34450     /* wait */
34451     #include "pthread.h"
34452     #include "sem.h"

34453     int
34454     main()
34455     {
34456         semaphore_t *semap;

34457         semap = semaphore_open("/tmp/semaphore");
34458         if (semap == NULL)
34459             exit(1);
34460         semaphore_wait(semap);
34461         semaphore_close(semap);
34462         return (0);
34463     }

```

34464 FUTURE DIRECTIONS

34465 None.

34466 SEE ALSO

34467 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*, *pthread_mutexattr_destroy()*, the
34468 Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

34469 CHANGE HISTORY

34470 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34471 Issue 6

34472 The *pthread_mutexattr_destroy()* and *pthread_mutexattr_init()* functions are marked as part of the
34473 Threads option.

34474 IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

34475 **NAME**

34476 pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling — get and set the
 34477 prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

34478 **SYNOPSIS**

34479 THR TPP #include <pthread.h>

```
34480 int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *
34481     restrict attr, int *restrict prioceiling);
34482 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
34483     int prioceiling);
```

34485 **DESCRIPTION**

34486 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions,
 34487 respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to
 34488 by *attr* which was previously created by the function *pthread_mutexattr_init()*.

34489 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of
 34490 *prioceiling* are within the maximum range of priorities defined by SCHED_FIFO.

34491 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum
 34492 priority level at which the critical section guarded by the mutex is executed. In order to avoid
 34493 priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal
 34494 to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are
 34495 within the maximum range of priorities defined under the SCHED_FIFO scheduling policy.

34496 **RETURN VALUE**

34497 Upon successful completion, the *pthread_mutexattr_getprioceiling()* and
 34498 *pthread_mutexattr_setprioceiling()* functions shall return zero; otherwise, an error number shall be
 34499 returned to indicate the error.

34500 **ERRORS**

34501 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions may fail if:

34502 [EINVAL] The value specified by *attr* or *prioceiling* is invalid.

34503 [EPERM] The caller does not have the privilege to perform the operation.

34504 These functions shall not return an error code of [EINTR].

34505 **EXAMPLES**

34506 None.

34507 **APPLICATION USAGE**

34508 None.

34509 **RATIONALE**

34510 None.

34511 **FUTURE DIRECTIONS**

34512 None.

34513 **SEE ALSO**

34514 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*, the Base Definitions volume of
 34515 IEEE Std 1003.1-2001, <pthread.h>

CHANGE HISTORY

34517 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34518 Marked as part of the Realtime Threads Feature Group.

Issue 6

34520 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions are marked as part of the Threads and Thread Priority Protection options.

34522 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection option.

34524 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol* argument.

34526 The **restrict** keyword is added to the *pthread_mutexattr_getprioceiling()* prototype for alignment with the ISO / IEC 9899:1999 standard.

34528 **NAME**

34529 pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol — get and set the protocol
 34530 attribute of the mutex attributes object (**REALTIME THREADS**)

34531 **SYNOPSIS**

34532 THR #include <pthread.h>

34533 TPP|TPI int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *
 34534 restrict attr, int *restrict protocol);
 34535 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
 34536 int protocol);
 34537

34538 **DESCRIPTION**

34539 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions, respectively,
 34540 shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was
 34541 previously created by the function *pthread_mutexattr_init()*.

34542 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of
 34543 *protocol* may be one of:

34544 PTHREAD_PRIO_NONE
 34545 TPI PTHREAD_PRIO_INHERIT
 34546 TPP PTHREAD_PRIO_PROTECT
 34547

34548 which are defined in the <pthread.h> header.

34549 When a thread owns a mutex with the PTHREAD_PRIO_NONE *protocol* attribute, its priority
 34550 and scheduling shall not be affected by its mutex ownership.

34551 TPI When a thread is blocking higher priority threads because of owning one or more mutexes with
 34552 the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of its priority or
 34553 the priority of the highest priority thread waiting on any of the mutexes owned by this thread
 34554 and initialized with this protocol.

34555 TPP When a thread owns one or more mutexes initialized with the PTHREAD_PRIO_PROTECT
 34556 protocol, it shall execute at the higher of its priority or the highest of the priority ceilings of all
 34557 the mutexes owned by this thread and initialized with this attribute, regardless of whether other
 34558 threads are blocked on any of these mutexes or not.

34559 While a thread is holding a mutex which has been initialized with the
 34560 PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol attributes, it shall not be
 34561 subject to being moved to the tail of the scheduling queue at its priority in the event that its
 34562 original priority is changed, such as by a call to *sched_setparam()*. Likewise, when a thread
 34563 unlocks a mutex that has been initialized with the PTHREAD_PRIO_INHERIT or
 34564 PTHREAD_PRIO_PROTECT protocol attributes, it shall not be subject to being moved to the tail
 34565 of the scheduling queue at its priority in the event that its original priority is changed.

34566 If a thread simultaneously owns several mutexes initialized with different protocols, it shall
 34567 execute at the highest of the priorities that it would have obtained by each of these protocols.

34568 TPI When a thread makes a call to *pthread_mutex_lock()*, the mutex was initialized with the protocol
 34569 attribute having the value PTHREAD_PRIO_INHERIT, when the calling thread is blocked
 34570 because the mutex is owned by another thread, that owner thread shall inherit the priority level
 34571 of the calling thread as long as it continues to own the mutex. The implementation shall update
 34572 its execution priority to the maximum of its assigned priority and all its inherited priorities.
 34573 Furthermore, if this owner thread itself becomes blocked on another mutex, the same priority

34574 inheritance effect shall be propagated to this other owner thread, in a recursive manner.

34575 RETURN VALUE

34576 Upon successful completion, the *pthread_mutexattr_getprotocol()* and
34577 *pthread_mutexattr_setprotocol()* functions shall return zero; otherwise, an error number shall be
34578 returned to indicate the error.

34579 ERRORS

34580 The *pthread_mutexattr_setprotocol()* function shall fail if:

34581 [ENOTSUP] The value specified by *protocol* is an unsupported value.

34582 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions may fail if:

34583 [EINVAL] The value specified by *attr* or *protocol* is invalid.

34584 [EPERM] The caller does not have the privilege to perform the operation.

34585 These functions shall not return an error code of [EINTR].

34586 EXAMPLES

34587 None.

34588 APPLICATION USAGE

34589 None.

34590 RATIONALE

34591 None.

34592 FUTURE DIRECTIONS

34593 None.

34594 SEE ALSO

34595 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*, the Base Definitions volume of
34596 IEEE Std 1003.1-2001, <pthread.h>

34597 CHANGE HISTORY

34598 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34599 Marked as part of the Realtime Threads Feature Group.

34600 Issue 6

34601 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions are marked as
34602 part of the Threads option and either the Thread Priority Protection or Thread Priority
34603 Inheritance options.

34604 The [ENOSYS] error condition has been removed as stubs need not be provided if an
34605 implementation does not support the Thread Priority Protection or Thread Priority Inheritance
34606 options.

34607 The **restrict** keyword is added to the *pthread_mutexattr_getprotocol()* prototype for alignment
34608 with the ISO / IEC 9899:1999 standard.

34609 **NAME**

34610 pthread_mutexattr_getpshared, pthread_mutexattr_setpshared — get and set the process-
 34611 shared attribute

34612 **SYNOPSIS**

34613 THR TSH #include <pthread.h>

```
34614 int pthread_mutexattr_getpshared(const pthread_mutexattr_t *
34615     restrict attr, int *restrict pshared);
34616 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
34617     int pshared);
34618
```

34619 **DESCRIPTION**

34620 The *pthread_mutexattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 34621 from the attributes object referenced by *attr*. The *pthread_mutexattr_setpshared()* function shall
 34622 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

34623 The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a mutex to be
 34624 operated upon by any thread that has access to the memory where the mutex is allocated, even if
 34625 the mutex is allocated in memory that is shared by multiple processes. If the *process-shared*
 34626 attribute is PTHREAD_PROCESS_PRIVATE, the mutex shall only be operated upon by threads
 34627 created within the same process as the thread that initialized the mutex; if threads of differing
 34628 processes attempt to operate on such a mutex, the behavior is undefined. The default value of
 34629 the attribute shall be PTHREAD_PROCESS_PRIVATE.

34630 **RETURN VALUE**

34631 Upon successful completion, *pthread_mutexattr_setpshared()* shall return zero; otherwise, an error
 34632 number shall be returned to indicate the error.

34633 Upon successful completion, *pthread_mutexattr_getpshared()* shall return zero and store the value
 34634 of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.
 34635 Otherwise, an error number shall be returned to indicate the error.

34636 **ERRORS**

34637 The *pthread_mutexattr_getpshared()* and *pthread_mutexattr_setpshared()* functions may fail if:

34638 [EINVAL] The value specified by *attr* is invalid.

34639 The *pthread_mutexattr_setpshared()* function may fail if:

34640 [EINVAL] The new value specified for the attribute is outside the range of legal values
 34641 for that attribute.

34642 These functions shall not return an error code of [EINTR].

34643 **EXAMPLES**

34644 None.

34645 **APPLICATION USAGE**

34646 None.

34647 **RATIONALE**

34648 None.

34649 **FUTURE DIRECTIONS**

34650 None.

34651 **SEE ALSO**

34652 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*, *pthread_mutexattr_destroy()*, the
34653 Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

34654 **CHANGE HISTORY**

34655 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34656 **Issue 6**

34657 The *pthread_mutexattr_getpshared()* and *pthread_mutexattr_setpshared()* functions are marked as
34658 part of the Threads and Thread Process-Shared Synchronization options.

34659 The **restrict** keyword is added to the *pthread_mutexattr_getpshared()* prototype for alignment
34660 with the ISO / IEC 9899:1999 standard.

34661 **NAME**

34662 pthread_mutexattr_gettype, pthread_mutexattr_settype — get and set the mutex type attribute

34663 **SYNOPSIS**34664 XSI

```
#include <pthread.h>
```

34665

```
int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,  
34666 int *restrict type);
```

34667

```
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

34668

34669 **DESCRIPTION**34670 The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions, respectively, shall get
34671 and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The
34672 default value of the *type* attribute is PTHREAD_MUTEX_DEFAULT.34673 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types
34674 include:

34675 PTHREAD_MUTEX_NORMAL

34676 This type of mutex does not detect deadlock. A thread attempting to relock this mutex
34677 without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a
34678 different thread results in undefined behavior. Attempting to unlock an unlocked mutex
34679 results in undefined behavior.

34680 PTHREAD_MUTEX_ERRORCHECK

34681 This type of mutex provides error checking. A thread attempting to relock this mutex
34682 without first unlocking it shall return with an error. A thread attempting to unlock a mutex
34683 which another thread has locked shall return with an error. A thread attempting to unlock
34684 an unlocked mutex shall return with an error.

34685 PTHREAD_MUTEX_RECURSIVE

34686 A thread attempting to relock this mutex without first unlocking it shall succeed in locking
34687 the mutex. The relocking deadlock which can occur with mutexes of type
34688 PTHREAD_MUTEX_NORMAL cannot occur with this type of mutex. Multiple locks of this
34689 mutex shall require the same number of unlocks to release the mutex before another thread
34690 can acquire the mutex. A thread attempting to unlock a mutex which another thread has
34691 locked shall return with an error. A thread attempting to unlock an unlocked mutex shall
34692 return with an error.

34693 PTHREAD_MUTEX_DEFAULT

34694 Attempting to recursively lock a mutex of this type results in undefined behavior.
34695 Attempting to unlock a mutex of this type which was not locked by the calling thread
34696 results in undefined behavior. Attempting to unlock a mutex of this type which is not
34697 locked results in undefined behavior. An implementation may map this mutex to one of the
34698 other mutex types.34699 **RETURN VALUE**34700 Upon successful completion, the *pthread_mutexattr_gettype()* function shall return zero and store
34701 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,
34702 an error shall be returned to indicate the error.34703 If successful, the *pthread_mutexattr_settype()* function shall return zero; otherwise, an error
34704 number shall be returned to indicate the error.

34705 **ERRORS**

34706 The *pthread_mutexattr_settype()* function shall fail if:

34707 [EINVAL] The value *type* is invalid.

34708 The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions may fail if:

34709 [EINVAL] The value specified by *attr* is invalid.

34710 These functions shall not return an error code of [EINTR].

34711 **EXAMPLES**

34712 None.

34713 **APPLICATION USAGE**

34714 It is advised that an application should not use a PTHREAD_MUTEX_RECURSIVE mutex with
34715 condition variables because the implicit unlock performed for a *pthread_cond_timedwait()* or
34716 *pthread_cond_wait()* may not actually release the mutex (if it had been locked multiple times). If
34717 this happens, no other thread can satisfy the condition of the predicate.

34718 **RATIONALE**

34719 None.

34720 **FUTURE DIRECTIONS**

34721 None.

34722 **SEE ALSO**

34723 *pthread_cond_timedwait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

34724 **CHANGE HISTORY**

34725 First released in Issue 5.

34726 **Issue 6**

34727 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for
34728 *pthread_mutexattr_gettype()* is updated so that the first argument is of type **const**
34729 **pthread_mutexattr_t***.

34730 The **restrict** keyword is added to the *pthread_mutexattr_gettype()* prototype for alignment with
34731 the ISO / IEC 9899:1999 standard.

34732 NAME

34733 pthread_mutexattr_init — initialize the mutex attributes object

34734 SYNOPSIS

34735 THR `#include <pthread.h>`

34736 `int pthread_mutexattr_init(pthread_mutexattr_t *attr);`

34737

34738 DESCRIPTION

34739 Refer to *pthread_mutexattr_destroy()*.

34740 **NAME**

34741 pthread_mutexattr_setprioceiling — set the prioceiling attribute of the mutex attributes object
34742 (**REALTIME THREADS**)

34743 **SYNOPSIS**

34744 THR TPP `#include <pthread.h>`

34745 `int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,`
34746 `int prioceiling);`
34747

34748 **DESCRIPTION**

34749 Refer to *pthread_mutexattr_getprioceiling()*.

34750 NAME

34751 pthread_mutexattr_setprotocol — set the protocol attribute of the mutex attributes object
34752 (**REALTIME THREADS**)

34753 SYNOPSIS

34754 THR `#include <pthread.h>`

34755 TPP|TPI `int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,`
34756 `int protocol);`

34757

34758 DESCRIPTION

34759 Refer to *pthread_mutexattr_getprotocol()*.

34760 **NAME**

34761 pthread_mutexattr_setpshared — set the process-shared attribute

34762 **SYNOPSIS**34763 THR TSH `#include <pthread.h>`

```
34764 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,  
34765 int pshared);
```

34766

34767 **DESCRIPTION**34768 Refer to *pthread_mutexattr_getpshared()*.

34769 NAME

34770 pthread_mutexattr_settype — set the mutex type attribute

34771 SYNOPSIS

34772 xSI `#include <pthread.h>`

34773 `int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);`

34774

34775 DESCRIPTION

34776 Refer to *pthread_mutexattr_gettype()*.

34777 **NAME**

34778 pthread_once — dynamic package initialization

34779 **SYNOPSIS**

```
34780 THR    #include <pthread.h>

34781        int pthread_once(pthread_once_t *once_control,
34782                          void (*init_routine)(void));
34783        pthread_once_t once_control = PTHREAD_ONCE_INIT;
34784
```

34785 **DESCRIPTION**

34786 The first call to *pthread_once()* by any thread in a process, with a given *once_control*, shall call the
 34787 *init_routine* with no arguments. Subsequent calls of *pthread_once()* with the same *once_control*
 34788 shall not call the *init_routine*. On return from *pthread_once()*, *init_routine* shall have completed.
 34789 The *once_control* parameter shall determine whether the associated initialization routine has
 34790 been called.

34791 The *pthread_once()* function is not a cancellation point. However, if *init_routine* is a cancellation
 34792 point and is canceled, the effect on *once_control* shall be as if *pthread_once()* was never called.

34793 The constant PTHREAD_ONCE_INIT is defined in the **<pthread.h>** header.

34794 The behavior of *pthread_once()* is undefined if *once_control* has automatic storage duration or is
 34795 not initialized by PTHREAD_ONCE_INIT.

34796 **RETURN VALUE**

34797 Upon successful completion, *pthread_once()* shall return zero; otherwise, an error number shall
 34798 be returned to indicate the error.

34799 **ERRORS**

34800 The *pthread_once()* function may fail if:

34801 [EINVAL] If either *once_control* or *init_routine* is invalid.

34802 The *pthread_once()* function shall not return an error code of [EINTR].

34803 **EXAMPLES**

34804 None.

34805 **APPLICATION USAGE**

34806 None.

34807 **RATIONALE**

34808 Some C libraries are designed for dynamic initialization. That is, the global initialization for the
 34809 library is performed when the first procedure in the library is called. In a single-threaded
 34810 program, this is normally implemented using a static variable whose value is checked on entry
 34811 to a routine, as follows:

```
34812 static int random_is_initialized = 0;
34813 extern int initialize_random();

34814 int random_function()
34815 {
34816     if (random_is_initialized == 0) {
34817         initialize_random();
34818         random_is_initialized = 1;
34819     }
34820     ... /* Operations performed after initialization. */
34821 }
```

34822 To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise,
34823 library initialization has to be accomplished by an explicit call to a library-exported initialization
34824 function prior to any use of the library.

34825 For dynamic library initialization in a multi-threaded process, a simple initialization flag is not
34826 sufficient; the flag needs to be protected against modification by multiple threads
34827 simultaneously calling into the library. Protecting the flag requires the use of a mutex; however,
34828 mutexes have to be initialized before they are used. Ensuring that the mutex is only initialized
34829 once requires a recursive solution to this problem.

34830 The use of *pthread_once()* not only supplies an implementation-guaranteed means of dynamic
34831 initialization, it provides an aid to the reliable construction of multi-threaded and realtime
34832 systems. The preceding example then becomes:

```
34833 #include <pthread.h>
34834 static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
34835 extern int initialize_random();

34836 int random_function()
34837 {
34838     (void) pthread_once(&random_is_initialized, initialize_random);
34839     ... /* Operations performed after initialization. */
34840 }
```

34841 Note that a **pthread_once_t** cannot be an array because some compilers do not accept the
34842 construct **&<array_name>**.

34843 FUTURE DIRECTIONS

34844 None.

34845 SEE ALSO

34846 The Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

34847 CHANGE HISTORY

34848 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34849 Issue 6

34850 The *pthread_once()* function is marked as part of the Threads option.

34851 The [EINVAL] error is added as a may fail case for if either argument is invalid.

34852 **NAME**

34853 pthread_rwlock_destroy, pthread_rwlock_init — destroy and initialize a read-write lock object

34854 **SYNOPSIS**34855 THR

```
#include <pthread.h>
```

```
34856 int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
34857 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
34858     const pthread_rwlockattr_t *restrict attr);
34859
```

34860 **DESCRIPTION**

34861 The *pthread_rwlock_destroy()* function shall destroy the read-write lock object referenced by
 34862 *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is
 34863 undefined until the lock is reinitialized by another call to *pthread_rwlock_init()*. An
 34864 implementation may cause *pthread_rwlock_destroy()* to set the object referenced by *rwlock* to an
 34865 invalid value. Results are undefined if *pthread_rwlock_destroy()* is called when any thread holds
 34866 *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

34867 The *pthread_rwlock_init()* function shall allocate any resources required to use the read-write
 34868 lock referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced
 34869 by *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same
 34870 as passing the address of a default read-write lock attributes object. Once initialized, the lock can
 34871 be used any number of times without being reinitialized. Results are undefined if
 34872 *pthread_rwlock_init()* is called specifying an already initialized read-write lock. Results are
 34873 undefined if a read-write lock is used without first being initialized.

34874 If the *pthread_rwlock_init()* function fails, *rwlock* shall not be initialized and the contents of *rwlock*
 34875 are undefined.

34876 Only the object referenced by *rwlock* may be used for performing synchronization. The result of
 34877 referring to copies of that object in calls to *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*,
 34878 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_tryrdlock()*,
 34879 *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*, or *pthread_rwlock_wrlock()* is undefined.

34880 **RETURN VALUE**

34881 If successful, the *pthread_rwlock_destroy()* and *pthread_rwlock_init()* functions shall return zero;
 34882 otherwise, an error number shall be returned to indicate the error.

34883 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed
 34884 immediately at the beginning of processing for the function and caused an error return prior to
 34885 modifying the state of the read-write lock specified by *rwlock*.

34886 **ERRORS**

34887 The *pthread_rwlock_destroy()* function may fail if:

34888 [EBUSY] The implementation has detected an attempt to destroy the object referenced
 34889 by *rwlock* while it is locked.

34890 [EINVAL] The value specified by *rwlock* is invalid.

34891 The *pthread_rwlock_init()* function shall fail if:

34892 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
 34893 another read-write lock.

34894 [ENOMEM] Insufficient memory exists to initialize the read-write lock.

34895 [EPERM] The caller does not have the privilege to perform the operation.

34896 The *pthread_rwlock_init()* function may fail if:

34897 [EBUSY] The implementation has detected an attempt to reinitialize the object
34898 referenced by *rwlock*, a previously initialized but not yet destroyed read-write
34899 lock.

34900 [EINVAL] The value specified by *attr* is invalid.

34901 These functions shall not return an error code of [EINTR].

34902 **EXAMPLES**

34903 None.

34904 **APPLICATION USAGE**

34905 Applications using these and related read-write lock functions may be subject to priority
34906 inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285,
34907 Priority Inversion.

34908 **RATIONALE**

34909 None.

34910 **FUTURE DIRECTIONS**

34911 None.

34912 **SEE ALSO**

34913 *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*,
34914 *pthread_rwlock_tryrdlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*,
34915 *pthread_rwlock_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

34916 **CHANGE HISTORY**

34917 First released in Issue 5.

34918 **Issue 6**

34919 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 34920 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
34921 now part of the Threads option (previously it was part of the Read-Write Locks option in
34922 IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also deleted
34923 from the SYNOPSIS.
- 34924 • The DESCRIPTION is updated as follows:
 - 34925 — It explicitly notes allocation of resources upon initialization of a read-write lock object.
 - 34926 — A paragraph is added specifying that copies of read-write lock objects may not be used.
- 34927 • An [EINVAL] error is added to the ERRORS section for *pthread_rwlock_init()*, indicating that
34928 the *rwlock* value is invalid.
- 34929 • The SEE ALSO section is updated.

34930 The **restrict** keyword is added to the *pthread_rwlock_init()* prototype for alignment with the
34931 ISO/IEC 9899:1999 standard.

34932 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION
34933 USAGE relating to priority inversion.

34934 NAME

34935 pthread_rwlock_rdlock, pthread_rwlock_tryrdlock — lock a read-write lock object for reading

34936 SYNOPSIS

34937 THR `#include <pthread.h>`

34938 `int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);`

34939 `int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);`

34940

34941 DESCRIPTION

34942 The *pthread_rwlock_rdlock()* function shall apply a read lock to the read-write lock referenced by
34943 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are
34944 no writers blocked on the lock.

34945 TPS If the Thread Execution Scheduling option is supported, and the threads involved in the lock are
34946 executing with the scheduling policies SCHED_FIFO or SCHED_RR, the calling thread shall not
34947 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on
34948 the lock; otherwise, the calling thread shall acquire the lock.

34949 TPS TSP If the Threads Execution Scheduling option is supported, and the threads involved in the lock
34950 are executing with the SCHED_SPORADIC scheduling policy, the calling thread shall not
34951 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on
34952 the lock; otherwise, the calling thread shall acquire the lock.

34953 If the Thread Execution Scheduling option is not supported, it is implementation-defined
34954 whether the calling thread acquires the lock when a writer does not hold the lock and there are
34955 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the
34956 read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the
34957 lock. The calling thread may deadlock if at the time the call is made it holds a write lock.

34958 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the
34959 *pthread_rwlock_rdlock()* function *n* times). If so, the application shall ensure that the thread
34960 performs matching unlocks (that is, it calls the *pthread_rwlock_unlock()* function *n* times).

34961 The maximum number of simultaneous read locks that an implementation guarantees can be
34962 applied to a read-write lock shall be implementation-defined. The *pthread_rwlock_rdlock()*
34963 function may fail if this maximum would be exceeded.

34964 The *pthread_rwlock_tryrdlock()* function shall apply a read lock as in the *pthread_rwlock_rdlock()*
34965 function, with the exception that the function shall fail if the equivalent *pthread_rwlock_rdlock()*
34966 call would have blocked the calling thread. In no case shall the *pthread_rwlock_tryrdlock()*
34967 function ever block; it always either acquires the lock or fails and returns immediately.

34968 Results are undefined if any of these functions are called with an uninitialized read-write lock.

34969 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the
34970 signal handler the thread resumes waiting for the read-write lock for reading as if it was not
34971 interrupted.

34972 RETURN VALUE

34973 If successful, the *pthread_rwlock_rdlock()* function shall return zero; otherwise, an error number
34974 shall be returned to indicate the error.

34975 The *pthread_rwlock_tryrdlock()* function shall return zero if the lock for reading on the read-write
34976 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to
34977 indicate the error.

34978 **ERRORS**

34979 The *pthread_rwlock_tryrdlock()* function shall fail if:

34980 [EBUSY] The read-write lock could not be acquired for reading because a writer holds
34981 the lock or a writer with the appropriate priority was blocked on it.

34982 The *pthread_rwlock_rdlock()* and *pthread_rwlock_tryrdlock()* functions may fail if:

34983 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
34984 object.

34985 [EAGAIN] The read lock could not be acquired because the maximum number of read
34986 locks for *rwlock* has been exceeded.

34987 The *pthread_rwlock_rdlock()* function may fail if:

34988 [EDEADLK] The current thread already owns the read-write lock for writing.

34989 These functions shall not return an error code of [EINTR].

34990 **EXAMPLES**

34991 None.

34992 **APPLICATION USAGE**

34993 Applications using these functions may be subject to priority inversion, as discussed in the Base
34994 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

34995 **RATIONALE**

34996 None.

34997 **FUTURE DIRECTIONS**

34998 None.

34999 **SEE ALSO**

35000 *pthread_rwlock_destroy()*, *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*,
35001 *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*, *pthread_rwlock_wrlock()*, the Base Definitions
35002 volume of IEEE Std 1003.1-2001, <**pthread.h**>

35003 **CHANGE HISTORY**

35004 First released in Issue 5.

35005 **Issue 6**

35006 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35007 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
35008 now part of the Threads option (previously it was part of the Read-Write Locks option in
35009 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35010 • The DESCRIPTION is updated as follows:
 - 35011 — Conditions under which writers have precedence over readers are specified.
 - 35012 — Failure of *pthread_rwlock_tryrdlock()* is clarified.
 - 35013 — A paragraph on the maximum number of read locks is added.
- 35014 • In the ERRORS sections, [EBUSY] is modified to take into account write priority, and
35015 [EDEADLK] is deleted as a *pthread_rwlock_tryrdlock()* error.
- 35016 • The SEE ALSO section is updated.

35017 NAME

35018 pthread_rwlock_timedrdlock — lock a read-write lock for reading

35019 SYNOPSIS

```
35020 THR TMO #include <pthread.h>
35021          #include <time.h>
```

```
35022          int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
35023                                         const struct timespec *restrict abs_timeout);
35024
```

35025 DESCRIPTION

35026 The *pthread_rwlock_timedrdlock()* function shall apply a read lock to the read-write lock
 35027 referenced by *rwlock* as in the *pthread_rwlock_rdlock()* function. However, if the lock cannot be
 35028 acquired without waiting for other threads to unlock the lock, this wait shall be terminated
 35029 when the specified timeout expires. The timeout shall expire when the absolute time specified
 35030 by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the
 35031 value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout*
 35032 has already been passed at the time of the call.

35033 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If
 35034 the Timers option is not supported, the timeout shall be based on the system clock as returned
 35035 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which
 35036 it is based. The **timespec** data type is defined in the `<time.h>` header. Under no circumstances
 35037 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the
 35038 *abs_timeout* parameter need not be checked if the lock can be immediately acquired.

35039 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-
 35040 write lock via a call to *pthread_rwlock_timedrdlock()*, upon return from the signal handler the
 35041 thread shall resume waiting for the lock as if it was not interrupted.

35042 The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*.
 35043 The results are undefined if this function is called with an uninitialized read-write lock.

35044 RETURN VALUE

35045 The *pthread_rwlock_timedrdlock()* function shall return zero if the lock for reading on the read-
 35046 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned
 35047 to indicate the error.

35048 ERRORS

35049 The *pthread_rwlock_timedrdlock()* function shall fail if:

35050 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35051 The *pthread_rwlock_timedrdlock()* function may fail if:

35052 [EAGAIN] The read lock could not be acquired because the maximum number of read
 35053 locks for lock would be exceeded.

35054 [EDEADLK] The calling thread already holds a write lock on *rwlock*.

35055 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
 35056 object, or the *abs_timeout* nanosecond value is less than zero or greater than or
 35057 equal to 1 000 million.

35058 This function shall not return an error code of [EINTR].

35059 EXAMPLES

35060 None.

35061 APPLICATION USAGE

35062 Applications using this function may be subject to priority inversion, as discussed in the Base
35063 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35064 The *pthread_rwlock_timedrdlock()* function is part of the Threads and Timeouts options and need
35065 not be provided on all implementations.

35066 RATIONALE

35067 None.

35068 FUTURE DIRECTIONS

35069 None.

35070 SEE ALSO

35071 *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*, *pthread_rwlock_timedwrlock()*,
35072 *pthread_rwlock_tryrdlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*,
35073 *pthread_rwlock_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**,
35074 **<time.h>**

35075 CHANGE HISTORY

35076 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35077 NAME

35078 pthread_rwlock_timedwrlock — lock a read-write lock for writing

35079 SYNOPSIS

```
35080 THR TMO #include <pthread.h>
35081          #include <time.h>
```

```
35082          int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rwlock,
35083                                         const struct timespec *restrict abs_timeout);
35084
```

35085 DESCRIPTION

35086 The *pthread_rwlock_timedwrlock()* function shall apply a write lock to the read-write lock
 35087 referenced by *rwlock* as in the *pthread_rwlock_wrlock()* function. However, if the lock cannot be
 35088 acquired without waiting for other threads to unlock the lock, this wait shall be terminated
 35089 when the specified timeout expires. The timeout shall expire when the absolute time specified
 35090 by *abs_timeout* passes, as measured by the clock on which timeouts are based (that is, when the
 35091 value of that clock equals or exceeds *abs_timeout*), or if the absolute time specified by *abs_timeout*
 35092 has already been passed at the time of the call.

35093 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If
 35094 the Timers option is not supported, the timeout shall be based on the system clock as returned
 35095 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which
 35096 it is based. The **timespec** data type is defined in the `<time.h>` header. Under no circumstances
 35097 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the
 35098 *abs_timeout* parameter need not be checked if the lock can be immediately acquired.

35099 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-
 35100 write lock via a call to *pthread_rwlock_timedwrlock()*, upon return from the signal handler the
 35101 thread shall resume waiting for the lock as if it was not interrupted.

35102 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The
 35103 results are undefined if this function is called with an uninitialized read-write lock.

35104 RETURN VALUE

35105 The *pthread_rwlock_timedwrlock()* function shall return zero if the lock for writing on the read-
 35106 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned
 35107 to indicate the error.

35108 ERRORS

35109 The *pthread_rwlock_timedwrlock()* function shall fail if:

35110 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35111 The *pthread_rwlock_timedwrlock()* function may fail if:

35112 [EDEADLK] The calling thread already holds the *rwlock*.

35113 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
 35114 object, or the *abs_timeout* nanosecond value is less than zero or greater than or
 35115 equal to 1 000 million.

35116 This function shall not return an error code of [EINTR].

35117 **EXAMPLES**

35118 None.

35119 **APPLICATION USAGE**

35120 Applications using this function may be subject to priority inversion, as discussed in the Base
35121 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35122 The *pthread_rwlock_timedwrlock()* function is part of the Threads and Timeouts options and need
35123 not be provided on all implementations.

35124 **RATIONALE**

35125 None.

35126 **FUTURE DIRECTIONS**

35127 None.

35128 **SEE ALSO**

35129 *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
35130 *pthread_rwlock_tryrdlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*,
35131 *pthread_rwlock_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**,
35132 **<time.h>**

35133 **CHANGE HISTORY**

35134 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35135 **NAME**

35136 pthread_rwlock_tryrdlock — lock a read-write lock object for reading

35137 **SYNOPSIS**

35138 THR `#include <pthread.h>`

35139 `int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);`

35140

35141 **DESCRIPTION**

35142 Refer to *pthread_rwlock_rdlock()*.

35143 **NAME**

35144 pthread_rwlock_trywrlock, pthread_rwlock_wrlock — lock a read-write lock object for writing

35145 **SYNOPSIS**

35146 THR #include <pthread.h>

35147 int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);

35148 int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);

35149

35150 **DESCRIPTION**

35151 The *pthread_rwlock_trywrlock()* function shall apply a write lock like the *pthread_rwlock_wrlock()*
 35152 function, with the exception that the function shall fail if any thread currently holds *rwlock* (for
 35153 reading or writing).

35154 The *pthread_rwlock_wrlock()* function shall apply a write lock to the read-write lock referenced
 35155 by *rwlock*. The calling thread acquires the write lock if no other thread (reader or writer) holds
 35156 the read-write lock *rwlock*. Otherwise, the thread shall block until it can acquire the lock. The
 35157 calling thread may deadlock if at the time the call is made it holds the read-write lock (whether a
 35158 read or write lock).

35159 Implementations may favor writers over readers to avoid writer starvation.

35160 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35161 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the
 35162 signal handler the thread resumes waiting for the read-write lock for writing as if it was not
 35163 interrupted.

35164 **RETURN VALUE**

35165 The *pthread_rwlock_trywrlock()* function shall return zero if the lock for writing on the read-write
 35166 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to
 35167 indicate the error.

35168 If successful, the *pthread_rwlock_wrlock()* function shall return zero; otherwise, an error number
 35169 shall be returned to indicate the error.

35170 **ERRORS**

35171 The *pthread_rwlock_trywrlock()* function shall fail if:

35172 [EBUSY] The read-write lock could not be acquired for writing because it was already
 35173 locked for reading or writing.

35174 The *pthread_rwlock_trywrlock()* and *pthread_rwlock_wrlock()* functions may fail if:

35175 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
 35176 object.

35177 The *pthread_rwlock_wrlock()* function may fail if:

35178 [EDEADLK] The current thread already owns the read-write lock for writing or reading.

35179 These functions shall not return an error code of [EINTR].

35180 EXAMPLES

35181 None.

35182 APPLICATION USAGE

35183 Applications using these functions may be subject to priority inversion, as discussed in the Base
35184 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35185 RATIONALE

35186 None.

35187 FUTURE DIRECTIONS

35188 None.

35189 SEE ALSO

35190 *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
35191 *pthread_rwlock_timedwrlock()*, *pthread_rwlock_tryrdlock()*, *pthread_rwlock_unlock()*, the Base
35192 Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

35193 CHANGE HISTORY

35194 First released in Issue 5.

35195 Issue 6

35196 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35197 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
35198 now part of the Threads option (previously it was part of the Read-Write Locks option in
35199 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35200 • The [EDEADLK] error is deleted as a *pthread_rwlock_trywrlock()* error.
- 35201 • The SEE ALSO section is updated.

35202 **NAME**

35203 pthread_rwlock_unlock — unlock a read-write lock object

35204 **SYNOPSIS**

35205 THR #include <pthread.h>

35206 int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);

35207

35208 **DESCRIPTION**

35209 The *pthread_rwlock_unlock()* function shall release a lock held on the read-write lock object
 35210 referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the
 35211 calling thread.

35212 If this function is called to release a read lock from the read-write lock object and there are other
 35213 read locks currently held on this read-write lock object, the read-write lock object remains in the
 35214 read locked state. If this function releases the last read lock for this read-write lock object, the
 35215 read-write lock object shall be put in the unlocked state with no owners.

35216 If this function is called to release a write lock for this read-write lock object, the read-write lock
 35217 object shall be put in the unlocked state.

35218 If there are threads blocked on the lock when it becomes available, the scheduling policy shall
 35219 TPS determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is
 35220 supported, when threads executing with the scheduling policies SCHED_FIFO, SCHED_RR, or
 35221 SCHED_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when
 35222 the lock becomes available. For equal priority threads, write locks shall take precedence over
 35223 read locks. If the Thread Execution Scheduling option is not supported, it is implementation-
 35224 defined whether write locks take precedence over read locks.

35225 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35226 **RETURN VALUE**

35227 If successful, the *pthread_rwlock_unlock()* function shall return zero; otherwise, an error number
 35228 shall be returned to indicate the error.

35229 **ERRORS**

35230 The *pthread_rwlock_unlock()* function may fail if:

35231 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
 35232 object.

35233 [EPERM] The current thread does not hold a lock on the read-write lock.

35234 The *pthread_rwlock_unlock()* function shall not return an error code of [EINTR].

35235 **EXAMPLES**

35236 None.

35237 **APPLICATION USAGE**

35238 None.

35239 **RATIONALE**

35240 None.

35241 **FUTURE DIRECTIONS**

35242 None.

35243 SEE ALSO

35244 *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
35245 *pthread_rwlock_timedwrlock()*, *pthread_rwlock_tryrdlock()*, *pthread_rwlock_trywrlock()*,
35246 *pthread_rwlock_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

35247 CHANGE HISTORY

35248 First released in Issue 5.

35249 Issue 6

35250 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35251 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
35252 now part of the Threads option (previously it was part of the Read-Write Locks option in
35253 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35254 • The DESCRIPTION is updated as follows:
 - 35255 — The conditions under which writers have precedence over readers are specified.
 - 35256 — The concept of read-write lock owner is deleted.
- 35257 • The SEE ALSO section is updated.

35258 NAME

35259 pthread_rwlock_wrlock — lock a read-write lock object for writing

35260 SYNOPSIS

35261 THR `#include <pthread.h>`

35262 `int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);`

35263

35264 DESCRIPTION

35265 Refer to *pthread_rwlock_trywrlock()*.

35266 **NAME**

35267 pthread_rwlockattr_destroy, pthread_rwlockattr_init — destroy and initialize the read-write
 35268 lock attributes object

35269 **SYNOPSIS**

35270 THR `#include <pthread.h>`

35271 `int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);`

35272 `int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);`

35273

35274 **DESCRIPTION**

35275 The *pthread_rwlockattr_destroy()* function shall destroy a read-write lock attributes object. A
 35276 destroyed *attr* attributes object can be reinitialized using *pthread_rwlockattr_init()*; the results of
 35277 otherwise referencing the object after it has been destroyed are undefined. An implementation
 35278 may cause *pthread_rwlockattr_destroy()* to set the object referenced by *attr* to an invalid value.

35279 The *pthread_rwlockattr_init()* function shall initialize a read-write lock attributes object *attr* with
 35280 the default value for all of the attributes defined by the implementation.

35281 Results are undefined if *pthread_rwlockattr_init()* is called specifying an already initialized *attr*
 35282 attributes object.

35283 After a read-write lock attributes object has been used to initialize one or more read-write locks,
 35284 any function affecting the attributes object (including destruction) shall not affect any previously
 35285 initialized read-write locks.

35286 **RETURN VALUE**

35287 If successful, the *pthread_rwlockattr_destroy()* and *pthread_rwlockattr_init()* functions shall return
 35288 zero; otherwise, an error number shall be returned to indicate the error.

35289 **ERRORS**

35290 The *pthread_rwlockattr_destroy()* function may fail if:

35291 [EINVAL] The value specified by *attr* is invalid.

35292 The *pthread_rwlockattr_init()* function shall fail if:

35293 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

35294 These functions shall not return an error code of [EINTR].

35295 **EXAMPLES**

35296 None.

35297 **APPLICATION USAGE**

35298 None.

35299 **RATIONALE**

35300 None.

35301 **FUTURE DIRECTIONS**

35302 None.

35303 **SEE ALSO**

35304 *pthread_rwlock_destroy()*, *pthread_rwlockattr_getpshared()*, *pthread_rwlockattr_setpshared()*, the
 35305 Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

35306 **CHANGE HISTORY**

35307 First released in Issue 5.

35308 **Issue 6**

35309 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35310 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
- 35311 now part of the Threads option (previously it was part of the Read-Write Locks option in
- 35312 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35313 • The SEE ALSO section is updated.

35314 **NAME**

35315 pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared — get and set the process-
 35316 shared attribute of the read-write lock attributes object

35317 **SYNOPSIS**

35318 THR TSH #include <pthread.h>

```
35319 int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *
35320     restrict attr, int *restrict pshared);
35321 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
35322     int pshared);
35323
```

35324 **DESCRIPTION**

35325 The *pthread_rwlockattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 35326 from the initialized attributes object referenced by *attr*. The *pthread_rwlockattr_setpshared()*
 35327 function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

35328 The *process-shared* attribute shall be set to PTHREAD_PROCESS_SHARED to permit a read-
 35329 write lock to be operated upon by any thread that has access to the memory where the read-
 35330 write lock is allocated, even if the read-write lock is allocated in memory that is shared by
 35331 multiple processes. If the *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the read-
 35332 write lock shall only be operated upon by threads created within the same process as the thread
 35333 that initialized the read-write lock; if threads of differing processes attempt to operate on such a
 35334 read-write lock, the behavior is undefined. The default value of the *process-shared* attribute shall
 35335 be PTHREAD_PROCESS_PRIVATE.

35336 Additional attributes, their default values, and the names of the associated functions to get and
 35337 set those attribute values are implementation-defined.

35338 **RETURN VALUE**

35339 Upon successful completion, the *pthread_rwlockattr_getpshared()* function shall return zero and
 35340 store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared*
 35341 parameter. Otherwise, an error number shall be returned to indicate the error.

35342 If successful, the *pthread_rwlockattr_setpshared()* function shall return zero; otherwise, an error
 35343 number shall be returned to indicate the error.

35344 **ERRORS**

35345 The *pthread_rwlockattr_getpshared()* and *pthread_rwlockattr_setpshared()* functions may fail if:

35346 [EINVAL] The value specified by *attr* is invalid.

35347 The *pthread_rwlockattr_setpshared()* function may fail if:

35348 [EINVAL] The new value specified for the attribute is outside the range of legal values
 35349 for that attribute.

35350 These functions shall not return an error code of [EINTR].

35351 **EXAMPLES**

35352 None.

35353 **APPLICATION USAGE**

35354 None.

35355 **RATIONALE**

35356 None.

35357 **FUTURE DIRECTIONS**

35358 None.

35359 **SEE ALSO**

35360 *pthread_rwlock_destroy()*, *pthread_rwlockattr_destroy()*, *pthread_rwlockattr_init()*, the Base
35361 Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

35362 **CHANGE HISTORY**

35363 First released in Issue 5.

35364 **Issue 6**

35365 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35366 • The margin code in the SYNOPSIS is changed to THR TSH to indicate that the functionality
35367 is now part of the Threads option (previously it was part of the Read-Write Locks option in
35368 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35369 • The DESCRIPTION notes that additional attributes are implementation-defined.
- 35370 • The SEE ALSO section is updated.

35371 The **restrict** keyword is added to the *pthread_rwlockattr_getpshared()* prototype for alignment
35372 with the ISO / IEC 9899:1999 standard.

35373 **NAME**

35374 pthread_rwlockattr_init — initialize the read-write lock attributes object

35375 **SYNOPSIS**

35376 THR #include <pthread.h>

35377 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);

35378

35379 **DESCRIPTION**35380 Refer to *pthread_rwlockattr_destroy()*.

35381 NAME

35382 pthread_rwlockattr_setpshared — set the process-shared attribute of the read-write lock
35383 attributes object

35384 SYNOPSIS

35385 THR TSH #include <pthread.h>

35386 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
35387 int pshared);

35388

35389 DESCRIPTION

35390 Refer to *pthread_rwlockattr_getpshared()*.

35391 **NAME**

35392 pthread_self — get the calling thread ID

35393 **SYNOPSIS**

35394 THR #include <pthread.h>

35395 pthread_t pthread_self(void);

35396

35397 **DESCRIPTION**35398 The *pthread_self()* function shall return the thread ID of the calling thread.35399 **RETURN VALUE**

35400 Refer to the DESCRIPTION.

35401 **ERRORS**

35402 No errors are defined.

35403 The *pthread_self()* function shall not return an error code of [EINTR].35404 **EXAMPLES**

35405 None.

35406 **APPLICATION USAGE**

35407 None.

35408 **RATIONALE**

35409 The *pthread_self()* function provides a capability similar to the *getpid()* function for processes
35410 and the rationale is the same: the creation call does not provide the thread ID to the created
35411 thread.

35412 **FUTURE DIRECTIONS**

35413 None.

35414 **SEE ALSO**

35415 *pthread_create()*, *pthread_equal()*, the Base Definitions volume of IEEE Std 1003.1-2001,
35416 <pthread.h>

35417 **CHANGE HISTORY**

35418 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

35419 **Issue 6**35420 The *pthread_self()* function is marked as part of the Threads option.

35421 **NAME**

35422 pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel — set cancelability state

35423 **SYNOPSIS**

35424 THR #include <pthread.h>

35425 int pthread_setcancelstate(int *state*, int **oldstate*);35426 int pthread_setcanceltype(int *type*, int **oldtype*);

35427 void pthread_testcancel(void);

35428

35429 **DESCRIPTION**

35430 The *pthread_setcancelstate()* function shall atomically both set the calling thread's cancelability
 35431 state to the indicated *state* and return the previous cancelability state at the location referenced
 35432 by *oldstate*. Legal values for *state* are PTHREAD_CANCEL_ENABLE and
 35433 PTHREAD_CANCEL_DISABLE.

35434 The *pthread_setcanceltype()* function shall atomically both set the calling thread's cancelability
 35435 type to the indicated *type* and return the previous cancelability type at the location referenced by
 35436 *oldtype*. Legal values for *type* are PTHREAD_CANCEL_DEFERRED and
 35437 PTHREAD_CANCEL_ASYNCHRONOUS.

35438 The cancelability state and type of any newly created threads, including the thread in which
 35439 *main()* was first invoked, shall be PTHREAD_CANCEL_ENABLE and
 35440 PTHREAD_CANCEL_DEFERRED respectively.

35441 The *pthread_testcancel()* function shall create a cancellation point in the calling thread. The
 35442 *pthread_testcancel()* function shall have no effect if cancelability is disabled.

35443 **RETURN VALUE**

35444 If successful, the *pthread_setcancelstate()* and *pthread_setcanceltype()* functions shall return zero;
 35445 otherwise, an error number shall be returned to indicate the error.

35446 **ERRORS**35447 The *pthread_setcancelstate()* function may fail if:

35448 [EINVAL] The specified *state* is not PTHREAD_CANCEL_ENABLE or
 35449 PTHREAD_CANCEL_DISABLE.

35450 The *pthread_setcanceltype()* function may fail if:

35451 [EINVAL] The specified *type* is not PTHREAD_CANCEL_DEFERRED or
 35452 PTHREAD_CANCEL_ASYNCHRONOUS.

35453 These functions shall not return an error code of [EINTR].

35454 **EXAMPLES**

35455 None.

35456 **APPLICATION USAGE**

35457 None.

35458 **RATIONALE**

35459 The *pthread_setcancelstate()* and *pthread_setcanceltype()* functions control the points at which a
 35460 thread may be asynchronously canceled. For cancellation control to be usable in modular
 35461 fashion, some rules need to be followed.

35462 An object can be considered to be a generalization of a procedure. It is a set of procedures and
 35463 global variables written as a unit and called by clients not known by the object. Objects may
 35464 depend on other objects.

35465 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On
35466 exit from an object, the cancelability state should always be restored to its value on entry to the
35467 object.

35468 This follows from a modularity argument: if the client of an object (or the client of an object that
35469 uses that object) has disabled cancelability, it is because the client does not want to be concerned
35470 about cleaning up if the thread is canceled while executing some sequence of actions. If an object
35471 is called in such a state and it enables cancelability and a cancellation request is pending for that
35472 thread, then the thread is canceled, contrary to the wish of the client that disabled.

35473 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry
35474 to an object. But as with the cancelability state, on exit from an object the cancelability type
35475 should always be restored to its value on entry to the object.

35476 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously
35477 cancelable.

35478 **FUTURE DIRECTIONS**

35479 None.

35480 **SEE ALSO**

35481 *pthread_cancel()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

35482 **CHANGE HISTORY**

35483 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

35484 **Issue 6**

35485 The *pthread_setcancelstate()*, *pthread_setcanceltype()*, and *pthread_testcancel()* functions are marked
35486 as part of the Threads option.

35487 NAME

35488 pthread_setconcurrency — set the level of concurrency

35489 SYNOPSIS

35490 xSI `#include <pthread.h>`

35491 `int pthread_setconcurrency(int new_level);`

35492

35493 DESCRIPTION

35494 Refer to *pthread_getconcurrency()*.

35495 **NAME**

35496 pthread_setschedparam — dynamic thread scheduling parameters access (**REALTIME**
35497 **THREADS**)

35498 **SYNOPSIS**

35499 THR TPS #include <pthread.h>

```
35500 int pthread_setschedparam(pthread_t thread, int policy,  
35501     const struct sched_param *param);  
35502
```

35503 **DESCRIPTION**

35504 Refer to *pthread_getschedparam()*.

35505 **NAME**

35506 pthread_setschedprio — dynamic thread scheduling parameters access (**REALTIME**
 35507 **THREADS**)

35508 **SYNOPSIS**

35509 THR TPS #include <pthread.h>

35510 int pthread_setschedprio(pthread_t thread, int prio);

35511

35512 **DESCRIPTION**

35513 The *pthread_setschedprio()* function shall set the scheduling priority for the thread whose thread
 35514 ID is given by *thread* to the value given by *prio*. See **Scheduling Policies** (on page 44) for a
 35515 description on how this function call affects the ordering of the thread in the thread list for its
 35516 new priority.

35517 If the *pthread_setschedprio()* function fails, the scheduling priority of the target thread shall not be
 35518 changed.

35519 **RETURN VALUE**

35520 If successful, the *pthread_setschedprio()* function shall return zero; otherwise, an error number
 35521 shall be returned to indicate the error.

35522 **ERRORS**

35523 The *pthread_setschedprio()* function may fail if:

35524 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.

35525 [ENOTSUP] An attempt was made to set the priority to an unsupported value.

35526 [EPERM] The caller does not have the appropriate permission to set the scheduling
 35527 policy of the specified thread.

35528 [EPERM] The implementation does not allow the application to modify the priority to
 35529 the value specified.

35530 [ESRCH] The value specified by *thread* does not refer to an existing thread.

35531 The *pthread_setschedprio()* function shall not return an error code of [EINTR].

35532 **EXAMPLES**

35533 None.

35534 **APPLICATION USAGE**

35535 None.

35536 **RATIONALE**

35537 The *pthread_setschedprio()* function provides a way for an application to temporarily raise its
 35538 priority and then lower it again, without having the undesired side effect of yielding to other
 35539 threads of the same priority. This is necessary if the application is to implement its own
 35540 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This
 35541 capability is especially important if the implementation does not support the Thread Priority
 35542 Protection or Thread Priority Inheritance options, but even if those options are supported it is
 35543 needed if the application is to bound priority inheritance for other resources, such as
 35544 semaphores.

35545 The standard developers considered that while it might be preferable conceptually to solve this
 35546 problem by modifying the specification of *pthread_setschedparam()*, it was too late to make such a
 35547 change, as there may be implementations that would need to be changed. Therefore, this new
 35548 function was introduced.

35549 **FUTURE DIRECTIONS**

35550 None.

35551 **SEE ALSO**

35552 **Scheduling Policies** (on page 44), *pthread_getschedparam()*, the Base Definitions volume of
35553 IEEE Std 1003.1-2001, <**pthread.h**>

35554 **CHANGE HISTORY**

35555 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

35556 **NAME**

35557 pthread_setspecific — thread-specific data management

35558 **SYNOPSIS**

35559 THR `#include <pthread.h>`

35560 `int pthread_setspecific(pthread_key_t key, const void *value);`

35561

35562 **DESCRIPTION**

35563 Refer to *pthread_getspecific()*.

35564 **NAME**

35565 pthread_sigmask, sigprocmask — examine and change blocked signals

35566 **SYNOPSIS**

35567 #include <signal.h>

```
35568 THR  int pthread_sigmask(int how, const sigset_t *restrict set,
35569                      sigset_t *restrict oset);
35570 CX    int sigprocmask(int how, const sigset_t *restrict set,
35571                      sigset_t *restrict oset);
35572
```

35573 **DESCRIPTION**

35574 THR The *pthread_sigmask()* function shall examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process. The function shall be equivalent to *sigprocmask()*, without the restriction that the call be made in a single-threaded process.

35577 In a single-threaded process, the *sigprocmask()* function shall examine or change (or both) the signal mask of the calling thread.

35579 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the currently blocked set.

35581 The argument *how* indicates the way in which the set is changed, and the application shall ensure it consists of one of the following values:

35583 SIG_BLOCK The resulting set shall be the union of the current set and the signal set pointed to by *set*.

35585 SIG_SETMASK The resulting set shall be the signal set pointed to by *set*.

35586 SIG_UNBLOCK The resulting set shall be the intersection of the current set and the complement of the signal set pointed to by *set*.

35588 If the argument *oset* is not a null pointer, the previous mask shall be stored in the location pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the process' signal mask shall be unchanged; thus the call can be used to enquire about currently blocked signals.

35592 If there are any pending unblocked signals after the call to *sigprocmask()*, at least one of those signals shall be delivered before the call to *sigprocmask()* returns.

35594 It is not possible to block those signals which cannot be ignored. This shall be enforced by the system without causing an error to be indicated.

35596 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked, the result is undefined, unless the signal was generated by the *kill()* function, the *sigqueue()* function, or the *raise()* function.

35599 If *sigprocmask()* fails, the thread's signal mask shall not be changed.

35600 The use of the *sigprocmask()* function is unspecified in a multi-threaded process.

35601 **RETURN VALUE**

35602 THR Upon successful completion *pthread_sigmask()* shall return 0; otherwise, it shall return the corresponding error number.

35604 Upon successful completion, *sigprocmask()* shall return 0; otherwise, -1 shall be returned, *errno* shall be set to indicate the error, and the process' signal mask shall be unchanged.

35606 **ERRORS**

35607 THR The `pthread_sigmask()` and `sigprocmask()` functions shall fail if:

35608 [EINVAL] The value of the *how* argument is not equal to one of the defined values.

35609 THR The `pthread_sigmask()` function shall not return an error code of [EINTR].

35610 **EXAMPLES**

35611 None.

35612 **APPLICATION USAGE**

35613 None.

35614 **RATIONALE**

35615 When a process' signal mask is changed in a signal-catching function that is installed by
 35616 `sigaction()`, the restoration of the signal mask on return from the signal-catching function
 35617 overrides that change (see `sigaction()`). If the signal-catching function was installed with
 35618 `signal()`, it is unspecified whether this occurs.

35619 See `kill()` for a discussion of the requirement on delivery of signals.

35620 **FUTURE DIRECTIONS**

35621 None.

35622 **SEE ALSO**

35623 `sigaction()`, `sigaddset()`, `sigdelset()`, `sigemptyset()`, `sigfillset()`, `sigismember()`, `sigpending()`,
 35624 `sigqueue()`, `sigsuspend()`, the Base Definitions volume of IEEE Std 1003.1-2001, <**signal.h**>

35625 **CHANGE HISTORY**

35626 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

35627 **Issue 5**

35628 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

35629 The `pthread_sigmask()` function is added for alignment with the POSIX Threads Extension.

35630 **Issue 6**

35631 The `pthread_sigmask()` function is marked as part of the Threads option.

35632 The SYNOPSIS for `sigprocmask()` is marked as a CX extension to note that the presence of this
 35633 function in the <**signal.h**> header is an extension to the ISO C standard.

35634 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 35635 • The DESCRIPTION is updated to explicitly state the functions which may generate the
 35636 signal.

35637 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

35638 The **restrict** keyword is added to the `pthread_sigmask()` and `sigprocmask()` prototypes for
 35639 alignment with the ISO / IEC 9899: 1999 standard.

35640 **NAME**

35641 pthread_spin_destroy, pthread_spin_init — destroy or initialize a spin lock object (**ADVANCED**
 35642 **REALTIME THREADS**)

35643 **SYNOPSIS**

35644 THR SPI #include <pthread.h>

```
35645 int pthread_spin_destroy(pthread_spinlock_t *lock);
35646 int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
35647
```

35648 **DESCRIPTION**

35649 The *pthread_spin_destroy()* function shall destroy the spin lock referenced by *lock* and release any
 35650 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is
 35651 reinitialized by another call to *pthread_spin_init()*. The results are undefined if
 35652 *pthread_spin_destroy()* is called when a thread holds the lock, or if this function is called with an
 35653 uninitialized thread spin lock.

35654 The *pthread_spin_init()* function shall allocate any resources required to use the spin lock
 35655 referenced by *lock* and initialize the lock to an unlocked state.

35656 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is
 35657 PTHREAD_PROCESS_SHARED, the implementation shall permit the spin lock to be operated
 35658 upon by any thread that has access to the memory where the spin lock is allocated, even if it is
 35659 allocated in memory that is shared by multiple processes.

35660 If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is
 35661 PTHREAD_PROCESS_PRIVATE, or if the option is not supported, the spin lock shall only be
 35662 operated upon by threads created within the same process as the thread that initialized the spin
 35663 lock. If threads of differing processes attempt to operate on such a spin lock, the behavior is
 35664 undefined.

35665 The results are undefined if *pthread_spin_init()* is called specifying an already initialized spin
 35666 lock. The results are undefined if a spin lock is used without first being initialized.

35667 If the *pthread_spin_init()* function fails, the lock is not initialized and the contents of *lock* are
 35668 undefined.

35669 Only the object referenced by *lock* may be used for performing synchronization.

35670 The result of referring to copies of that object in calls to *pthread_spin_destroy()*,
 35671 *pthread_spin_lock()*, *pthread_spin_trylock()*, or *pthread_spin_unlock()* is undefined.

35672 **RETURN VALUE**

35673 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 35674 be returned to indicate the error.

35675 **ERRORS**

35676 These functions may fail if:

35677 [EBUSY] The implementation has detected an attempt to initialize or destroy a spin
 35678 lock while it is in use (for example, while being used in a *pthread_spin_lock()*
 35679 call) by another thread.

35680 [EINVAL] The value specified by *lock* is invalid.

35681 The *pthread_spin_init()* function shall fail if:

35682 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

35683 [ENOMEM] Insufficient memory exists to initialize the lock.

35684 These functions shall not return an error code of [EINTR].

35685 **EXAMPLES**

35686 None.

35687 **APPLICATION USAGE**

35688 The *pthread_spin_destroy()* and *pthread_spin_init()* functions are part of the Spin Locks option
35689 and need not be provided on all implementations.

35690 **RATIONALE**

35691 None.

35692 **FUTURE DIRECTIONS**

35693 None.

35694 **SEE ALSO**

35695 *pthread_spin_lock()*, *pthread_spin_unlock()*, the Base Definitions volume of IEEE Std 1003.1-2001,
35696 <pthread.h>

35697 **CHANGE HISTORY**

35698 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35699 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

35700 **NAME**

35701 pthread_spin_lock, pthread_spin_trylock — lock a spin lock object (**ADVANCED REALTIME**
 35702 **THREADS**)

35703 **SYNOPSIS**

35704 THR SPI #include <pthread.h>

```
35705 int pthread_spin_lock(pthread_spinlock_t *lock);
35706 int pthread_spin_trylock(pthread_spinlock_t *lock);
35707
```

35708 **DESCRIPTION**

35709 The *pthread_spin_lock()* function shall lock the spin lock referenced by *lock*. The calling thread
 35710 shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is,
 35711 shall not return from the *pthread_spin_lock()* call) until the lock becomes available. The results
 35712 are undefined if the calling thread holds the lock at the time the call is made. The
 35713 *pthread_spin_trylock()* function shall lock the spin lock referenced by *lock* if it is not held by any
 35714 thread. Otherwise, the function shall fail.

35715 The results are undefined if any of these functions is called with an uninitialized spin lock.

35716 **RETURN VALUE**

35717 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 35718 be returned to indicate the error.

35719 **ERRORS**

35720 These functions may fail if:

35721 [EINVAL] The value specified by *lock* does not refer to an initialized spin lock object.

35722 The *pthread_spin_lock()* function may fail if:

35723 [EDEADLK] The calling thread already holds the lock.

35724 The *pthread_spin_trylock()* function shall fail if:

35725 [EBUSY] A thread currently holds the lock.

35726 These functions shall not return an error code of [EINTR].

35727 **EXAMPLES**

35728 None.

35729 **APPLICATION USAGE**

35730 Applications using this function may be subject to priority inversion, as discussed in the Base
 35731 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35732 The *pthread_spin_lock()* and *pthread_spin_trylock()* functions are part of the Spin Locks option
 35733 and need not be provided on all implementations.

35734 **RATIONALE**

35735 None.

35736 **FUTURE DIRECTIONS**

35737 None.

35738 **SEE ALSO**

35739 *pthread_spin_destroy()*, *pthread_spin_unlock()*, the Base Definitions volume of
 35740 IEEE Std 1003.1-2001, <pthread.h>

35741 **CHANGE HISTORY**

35742 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35743 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

35744 **NAME**35745 pthread_spin_unlock — unlock a spin lock object (**ADVANCED REALTIME THREADS**)35746 **SYNOPSIS**

35747 THR SPI #include <pthread.h>

35748 int pthread_spin_unlock(pthread_spinlock_t *lock);

35749

35750 **DESCRIPTION**

35751 The *pthread_spin_unlock()* function shall release the spin lock referenced by *lock* which was
 35752 locked via the *pthread_spin_lock()* or *pthread_spin_trylock()* functions. The results are undefined if
 35753 the lock is not held by the calling thread. If there are threads spinning on the lock when
 35754 *pthread_spin_unlock()* is called, the lock becomes available and an unspecified spinning thread
 35755 shall acquire the lock.

35756 The results are undefined if this function is called with an uninitialized thread spin lock.

35757 **RETURN VALUE**

35758 Upon successful completion, the *pthread_spin_unlock()* function shall return zero; otherwise, an
 35759 error number shall be returned to indicate the error.

35760 **ERRORS**35761 The *pthread_spin_unlock()* function may fail if:

35762 [EINVAL] An invalid argument was specified.

35763 [EPERM] The calling thread does not hold the lock.

35764 This function shall not return an error code of [EINTR].

35765 **EXAMPLES**

35766 None.

35767 **APPLICATION USAGE**

35768 The *pthread_spin_unlock()* function is part of the Spin Locks option and need not be provided on
 35769 all implementations.

35770 **RATIONALE**

35771 None.

35772 **FUTURE DIRECTIONS**

35773 None.

35774 **SEE ALSO**

35775 *pthread_spin_destroy()*, *pthread_spin_lock()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 35776 <pthread.h>

35777 **CHANGE HISTORY**

35778 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35779 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

35780 **NAME**

35781 pthread_testcancel — set cancelability state

35782 **SYNOPSIS**

35783 THR `#include <pthread.h>`

35784 `void pthread_testcancel(void);`

35785

35786 **DESCRIPTION**

35787 Refer to *pthread_setcancelstate()*.

35788 **NAME**

35789 ptsname — get name of the slave pseudo-terminal device

35790 **SYNOPSIS**

35791 xSI #include <stdlib.h>

35792 char *ptsname(int *fildes*);

35793

35794 **DESCRIPTION**

35795 The *ptsname()* function shall return the name of the slave pseudo-terminal device associated
35796 with a master pseudo-terminal device. The *fildes* argument is a file descriptor that refers to the
35797 master device. The *ptsname()* function shall return a pointer to a string containing the pathname
35798 of the corresponding slave device.

35799 The *ptsname()* function need not be reentrant. A function that is not required to be reentrant is
35800 not required to be thread-safe.

35801 **RETURN VALUE**

35802 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the
35803 pseudo-terminal slave device. Upon failure, *ptsname()* shall return a null pointer. This could
35804 occur if *fildes* is an invalid file descriptor or if the slave device name does not exist in the file
35805 system.

35806 **ERRORS**

35807 No errors are defined.

35808 **EXAMPLES**

35809 None.

35810 **APPLICATION USAGE**35811 The value returned may point to a static data area that is overwritten by each call to *ptsname()*.35812 **RATIONALE**

35813 None.

35814 **FUTURE DIRECTIONS**

35815 None.

35816 **SEE ALSO**

35817 *grantpt()*, *open()*, *ttyname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-2001,
35818 <stdlib.h>

35819 **CHANGE HISTORY**

35820 First released in Issue 4, Version 2.

35821 **Issue 5**

35822 Moved from X/OPEN UNIX extension to BASE.

35823 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

35824 NAME

35825 putc — put a byte on a stream

35826 SYNOPSIS

35827 #include <stdio.h>

35828 int putc(int *c*, FILE **stream*);

35829 DESCRIPTION

35830 cx The functionality described on this reference page is aligned with the ISO C standard. Any
35831 conflict between the requirements described here and the ISO C standard is unintentional. This
35832 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

35833 The *putc()* function shall be equivalent to *fputc()*, except that if it is implemented as a macro it
35834 may evaluate *stream* more than once, so the argument should never be an expression with side
35835 effects.

35836 RETURN VALUE

35837 Refer to *fputc()*.

35838 ERRORS

35839 Refer to *fputc()*.

35840 EXAMPLES

35841 None.

35842 APPLICATION USAGE

35843 Since it may be implemented as a macro, *putc()* may treat a *stream* argument with side effects
35844 incorrectly. In particular, *putc(c,*f++)* does not necessarily work correctly. Therefore, use of this
35845 function is not recommended in such situations; *fputc()* should be used instead.

35846 RATIONALE

35847 None.

35848 FUTURE DIRECTIONS

35849 None.

35850 SEE ALSO

35851 *fputc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

35852 CHANGE HISTORY

35853 First released in Issue 1. Derived from Issue 1 of the SVID.

35854 **NAME**

35855 putc_unlocked — stdio with explicit client locking

35856 **SYNOPSIS**

35857 TSF #include <stdio.h>

35858 int putc_unlocked(int *c*, FILE **stream*);

35859

35860 **DESCRIPTION**35861 Refer to *getc_unlocked()*.

35862 NAME

35863 putchar — put a byte on a stdout stream

35864 SYNOPSIS

35865 #include <stdio.h>

35866 int putchar(int c);

35867 DESCRIPTION

35868 cx The functionality described on this reference page is aligned with the ISO C standard. Any
35869 conflict between the requirements described here and the ISO C standard is unintentional. This
35870 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

35871 The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.

35872 RETURN VALUE

35873 Refer to *fputc()*.

35874 ERRORS

35875 Refer to *fputc()*.

35876 EXAMPLES

35877 None.

35878 APPLICATION USAGE

35879 None.

35880 RATIONALE

35881 None.

35882 FUTURE DIRECTIONS

35883 None.

35884 SEE ALSO

35885 *putc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>

35886 CHANGE HISTORY

35887 First released in Issue 1. Derived from Issue 1 of the SVID.

35888 **NAME**

35889 putchar_unlocked — stdio with explicit client locking

35890 **SYNOPSIS**35891 TSF `#include <stdio.h>`35892 `int putchar_unlocked(int c);`

35893

35894 **DESCRIPTION**35895 Refer to *getc_unlocked()*.

35896 **NAME**

35897 putenv — change or add a value to an environment

35898 **SYNOPSIS**35899 xSI `#include <stdlib.h>`35900 `int putenv(char *string);`

35901

35902 **DESCRIPTION**

35903 The *putenv()* function shall use the *string* argument to set environment variable values. The
 35904 *string* argument should point to a string of the form "*name=value*". The *putenv()* function shall
 35905 make the value of the environment variable *name* equal to *value* by altering an existing variable
 35906 or creating a new one. In either case, the string pointed to by *string* shall become part of the
 35907 environment, so altering the string shall change the environment. The space used by *string* is no
 35908 longer used once a new string which defines *name* is passed to *putenv()*.

35909 The *putenv()* function need not be reentrant. A function that is not required to be reentrant is not
 35910 required to be thread-safe.

35911 **RETURN VALUE**

35912 Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value
 35913 and set *errno* to indicate the error.

35914 **ERRORS**35915 The *putenv()* function may fail if:

35916 [ENOMEM] Insufficient memory was available.

35917 **EXAMPLES**35918 **Changing the Value of an Environment Variable**

35919 The following example changes the value of the *HOME* environment variable to the value
 35920 */usr/home*.

```
35921 #include <stdlib.h>
35922 ...
35923 static char *var = "HOME=/usr/home";
35924 int ret;
35925 ret = putenv(var);
```

35926 **APPLICATION USAGE**

35927 The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in
 35928 conjunction with *getenv()*.

35929 See *exec*, for restrictions on changing the environment in multi-threaded applications.

35930 This routine may use *malloc()* to enlarge the environment.

35931 A potential error is to call *putenv()* with an automatic variable as the argument, then return from
 35932 the calling function while *string* is still part of the environment.

35933 The *setenv()* function is preferred over this function.

35934 **RATIONALE**

35935 The standard developers noted that *putenv()* is the only function available to add to the
 35936 environment without permitting memory leaks.

35937 **FUTURE DIRECTIONS**

35938 None.

35939 **SEE ALSO**35940 *exec*, *getenv()*, *malloc()*, *setenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>35941 **CHANGE HISTORY**

35942 First released in Issue 1. Derived from Issue 1 of the SVID.

35943 **Issue 5**35944 The type of the argument to this function is changed from **const char *** to **char ***. This was
35945 indicated as a FUTURE DIRECTION in previous issues.

35946 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

35947 **Issue 6**35948 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the
35949 DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to
35950 *exec*. |

35951 **NAME**35952 putmsg, putpmsg — send a message on a STREAM (**STREAMS**)35953 **SYNOPSIS**

35954 xSR #include <stropts.h>

```

35955 int putmsg(int fildev, const struct strbuf *ctlptr,
35956           const struct strbuf *dataptr, int flags);
35957 int putpmsg(int fildev, const struct strbuf *ctlptr,
35958            const struct strbuf *dataptr, int band, int flags);
35959 
```

35960 **DESCRIPTION**

35961 The *putmsg()* function shall create a message from a process buffer(s) and send the message to a
 35962 STREAMS file. The message may contain either a data part, a control part, or both. The data and
 35963 control parts are distinguished by placement in separate buffers, as described below. The
 35964 semantics of each part are defined by the STREAMS module that receives the message.

35965 The *putpmsg()* function is equivalent to *putmsg()*, except that the process can send messages in
 35966 different priority bands. Except where noted, all requirements on *putmsg()* also pertain to
 35967 *putpmsg()*.

35968 The *fildev* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and
 35969 *dataptr* arguments each point to a **strbuf** structure.

35970 The *ctlptr* argument points to the structure describing the control part, if any, to be included in
 35971 the message. The *buf* member in the **strbuf** structure points to the buffer where the control
 35972 information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen*
 35973 member is not used by *putmsg()*. In a similar manner, the argument *dataptr* specifies the data, if
 35974 any, to be included in the message. The *flags* argument indicates what type of message should be
 35975 sent and is described further below.

35976 To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer
 35977 and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the
 35978 application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part
 35979 shall be sent if either *dataptr(ctlptr)* is a null pointer or the *len* member of *dataptr(ctlptr)* is set to
 35980 -1.

35981 For *putmsg()*, if a control part is specified and *flags* is set to RS_HIPRI, a high priority message
 35982 shall be sent. If no control part is specified, and *flags* is set to RS_HIPRI, *putmsg()* shall fail and
 35983 set *errno* to [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) shall be
 35984 sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be
 35985 sent and 0 shall be returned.

35986 For *putpmsg()*, the flags are different. The *flags* argument is a bitmask with the following
 35987 mutually-exclusive flags defined: MSG_HIPRI and MSG_BAND. If *flags* is set to 0, *putpmsg()*
 35988 shall fail and set *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG_HIPRI
 35989 and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG_HIPRI and either
 35990 no control part is specified or *band* is set to a non-zero value, *putpmsg()* shall fail and set *errno* to
 35991 [EINVAL]. If *flags* is set to MSG_BAND, then a message shall be sent in the priority band
 35992 specified by *band*. If a control part and data part are not specified and *flags* is set to MSG_BAND,
 35993 no message shall be sent and 0 shall be returned.

35994 The *putmsg()* function shall block if the STREAM write queue is full due to internal flow control
 35995 conditions, with the following exceptions:

- 35996 • For high-priority messages, *putmsg()* shall not block on this condition and continues
 35997 processing the message.

35998 • For other messages, *putmsg()* shall not block but shall fail when the write queue is full and
 35999 O_NONBLOCK is set.

36000 The *putmsg()* function shall also block, unless prevented by lack of internal resources, while
 36001 waiting for the availability of message blocks in the STREAM, regardless of priority or whether
 36002 O_NONBLOCK has been specified. No partial message shall be sent.

36003 RETURN VALUE

36004 Upon successful completion, *putmsg()* and *putpmsg()* shall return 0; otherwise, they shall return
 36005 −1 and set *errno* to indicate the error.

36006 ERRORS

36007 The *putmsg()* and *putpmsg()* functions shall fail if:

36008 [EAGAIN] A non-priority message was specified, the O_NONBLOCK flag is set, and the
 36009 STREAM write queue is full due to internal flow control conditions; or buffers
 36010 could not be allocated for the message that was to be created.

36011 [EBADF] *fildes* is not a valid file descriptor open for writing.

36012 [EINTR] A signal was caught during *putmsg()*.

36013 [EINVAL] An undefined value is specified in *flags*, or *flags* is set to RS_HIPRI or
 36014 MSG_HIPRI and no control part is supplied, or the STREAM or multiplexer
 36015 referenced by *fildes* is linked (directly or indirectly) downstream from a
 36016 multiplexer, or *flags* is set to MSG_HIPRI and *band* is non-zero (for *putpmsg()*
 36017 only).

36018 [ENOSR] Buffers could not be allocated for the message that was to be created due to
 36019 insufficient STREAMS memory resources.

36020 [ENOSTR] A STREAM is not associated with *fildes*.

36021 [ENXIO] A hangup condition was generated downstream for the specified STREAM.

36022 [EPIPE] or [EIO] The *fildes* argument refers to a STREAMS-based pipe and the other end of the
 36023 pipe is closed. A SIGPIPE signal is generated for the calling thread.

36024 [ERANGE] The size of the data part of the message does not fall within the range
 36025 specified by the maximum and minimum packet sizes of the topmost
 36026 STREAM module. This value is also returned if the control part of the message
 36027 is larger than the maximum configured size of the control part of a message,
 36028 or if the data part of a message is larger than the maximum configured size of
 36029 the data part of a message.

36030 In addition, *putmsg()* and *putpmsg()* shall fail if the STREAM head had processed an
 36031 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of
 36032 *putmsg()* or *putpmsg()*, but reflects the prior error.

36033 **EXAMPLES**36034 **Sending a High-Priority Message**

36035 The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg()* does the
 36036 following:

- 36037 1. Creates a high-priority message with a control part and a data part, using the buffers
 36038 pointed to by *ctrlbuf* and *databuf*, respectively.
- 36039 2. Sends the message to the STREAMS file identified by *fd*.

```

36040 #include <stropts.h>
36041 #include <string.h>
36042 ...
36043 int fd;
36044 char *ctrlbuf = "This is the control part";
36045 char *databuf = "This is the data part";
36046 struct strbuf ctrl;
36047 struct strbuf data;
36048 int ret;

36049 ctrl.buf = ctrlbuf;
36050 ctrl.len = strlen(ctrlbuf);

36051 data.buf = databuf;
36052 data.len = strlen(databuf);

36053 ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

36054 **Using putpmsg()**

36055 This example has the same effect as the previous example. In this example, however, the
 36056 *putpmsg()* function creates and sends the message to the STREAMS file.

```

36057 #include <stropts.h>
36058 #include <string.h>
36059 ...
36060 int fd;
36061 char *ctrlbuf = "This is the control part";
36062 char *databuf = "This is the data part";
36063 struct strbuf ctrl;
36064 struct strbuf data;
36065 int ret;

36066 ctrl.buf = ctrlbuf;
36067 ctrl.len = strlen(ctrlbuf);

36068 data.buf = databuf;
36069 data.len = strlen(databuf);

36070 ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

36071 **APPLICATION USAGE**

36072 None.

36073 **RATIONALE**

36074 None.

36075 **FUTURE DIRECTIONS**

36076 None.

36077 **SEE ALSO**

36078 Section 2.6 (on page 38), *getmsg()*, *poll()*, *read()*, *write()*, the Base Definitions volume of
36079 IEEE Std 1003.1-2001, <**stropts.h**>

36080 **CHANGE HISTORY**

36081 First released in Issue 4, Version 2.

36082 **Issue 5**

36083 Moved from X/OPEN UNIX extension to BASE.

36084 The following text is removed from the DESCRIPTION: “The STREAM head guarantees that the
36085 control part of a message generated by *putmsg()* is at least 64 bytes in length”.

36086 **Issue 6**

36087 This function is marked as part of the XSI STREAMS Option Group.

36088 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

36089 **NAME**

36090 puts — put a string on standard output

36091 **SYNOPSIS**

36092 #include <stdio.h>

36093 int puts(const char *s);

36094 **DESCRIPTION**

36095 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 36096 conflict between the requirements described here and the ISO C standard is unintentional. This
 36097 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36098 The *puts()* function shall write the string pointed to by *s*, followed by a <newline>, to the
 36099 standard output stream *stdout*. The terminating null byte shall not be written.

36100 CX The *st_ctime* and *st_mtime* fields of the file shall be marked for update between the successful
 36101 execution of *puts()* and the next successful completion of a call to *fflush()* or *fclose()* on the same
 36102 stream or a call to *exit()* or *abort()*.

36103 **RETURN VALUE**

36104 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall
 36105 CX return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

36106 **ERRORS**36107 Refer to *fputc()*.36108 **EXAMPLES**36109 **Printing to Standard Output**

36110 The following example gets the current time, converts it to a string using *localtime()* and
 36111 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to
 36112 an event for which it is waiting.

```

36113 #include <time.h>
36114 #include <stdio.h>
36115 ...
36116 time_t now;
36117 int minutes_to_event;
36118 ...
36119 time(&now);
36120 printf("The time is ");
36121 puts(asctime(localtime(&now)));
36122 printf("There are %d minutes to the event.\n",
36123        minutes_to_event);
36124 ...

```

36125 **APPLICATION USAGE**36126 The *puts()* function appends a <newline>, while *fputs()* does not.36127 **RATIONALE**

36128 None.

36129 **FUTURE DIRECTIONS**

36130 None.

36131 **SEE ALSO**

36132 *fopen()*, *fputs()*, *putc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

36133 **CHANGE HISTORY**

36134 First released in Issue 1. Derived from Issue 1 of the SVID.

36135 **Issue 6**

36136 Extensions beyond the ISO C standard are marked.

36137 NAME

36138 pututxline — put an entry into the user accounting database

36139 SYNOPSIS

36140 xSI #include <utmpx.h>

36141 struct utmpx *pututxline(const struct utmpx *utmpx);

36142

36143 DESCRIPTION

36144 Refer to *endutxent()*.

36145 **NAME**

36146 putwc — put a wide character on a stream

36147 **SYNOPSIS**

36148 #include <stdio.h>

36149 #include <wchar.h>

36150 wint_t putwc(wchar_t wc, FILE *stream);

36151 **DESCRIPTION**

36152 cx The functionality described on this reference page is aligned with the ISO C standard. Any
36153 conflict between the requirements described here and the ISO C standard is unintentional. This
36154 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36155 The *putwc()* function shall be equivalent to *fputwc()*, except that if it is implemented as a macro
36156 it may evaluate *stream* more than once, so the argument should never be an expression with side
36157 effects.

36158 **RETURN VALUE**36159 Refer to *fputwc()*.36160 **ERRORS**36161 Refer to *fputwc()*.36162 **EXAMPLES**

36163 None.

36164 **APPLICATION USAGE**

36165 Since it may be implemented as a macro, *putwc()* may treat a *stream* argument with side effects
36166 incorrectly. In particular, *putwc(wc,*f++)* need not work correctly. Therefore, use of this function
36167 is not recommended; *fputwc()* should be used instead.

36168 **RATIONALE**

36169 None.

36170 **FUTURE DIRECTIONS**

36171 None.

36172 **SEE ALSO**36173 *fputwc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>, <wchar.h>36174 **CHANGE HISTORY**

36175 First released as a World-wide Portability Interface in Issue 4.

36176 **Issue 5**

36177 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
36178 is changed from **wint_t** to **wchar_t**.

36179 The Optional Header (OH) marking is removed from <stdio.h>.

36180 **NAME**

36181 putwchar — put a wide character on a stdout stream

36182 **SYNOPSIS**

36183 #include <wchar.h>

36184 wint_t putwchar(wchar_t wc);

36185 **DESCRIPTION**

36186 cx The functionality described on this reference page is aligned with the ISO C standard. Any
36187 conflict between the requirements described here and the ISO C standard is unintentional. This
36188 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36189 The function call *putwchar(wc)* shall be equivalent to *putwc(wc,stdout)*.

36190 **RETURN VALUE**

36191 Refer to *fputwc()*.

36192 **ERRORS**

36193 Refer to *fputwc()*.

36194 **EXAMPLES**

36195 None.

36196 **APPLICATION USAGE**

36197 None.

36198 **RATIONALE**

36199 None.

36200 **FUTURE DIRECTIONS**

36201 None.

36202 **SEE ALSO**

36203 *fputwc()*, *putwc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>

36204 **CHANGE HISTORY**

36205 First released in Issue 4.

36206 **Issue 5**

36207 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
36208 is changed from **wint_t** to **wchar_t**.

36209 **NAME**

36210 pwrite — write on a file

36211 **SYNOPSIS**

36212 #include <unistd.h>

```
36213 xSI      ssize_t pwrite(int fildes, const void *buf, size_t nbyte,  
36214           off_t offset);
```

36215

36216 **DESCRIPTION**36217 Refer to *write()*.

36218 **NAME**

36219 qsort — sort a table of data

36220 **SYNOPSIS**

36221 #include <stdlib.h>

```
36222       void qsort(void *base, size_t nel, size_t width,
36223                   int (*compar)(const void *, const void *));
```

36224 **DESCRIPTION**

36225 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 36226 conflict between the requirements described here and the ISO C standard is unintentional. This
 36227 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36228 The *qsort()* function shall sort an array of *nel* objects, the initial element of which is pointed to by
 36229 *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has
 36230 the value zero, the comparison function pointed to by *compar* shall not be called and no
 36231 rearrangement shall take place.

36232 The application shall ensure that the comparison function pointed to by *compar* does not alter the
 36233 contents of the array. The implementation may reorder elements of the array between calls to the
 36234 comparison function, but shall not alter the contents of any individual element.

36235 When the same objects (consisting of *width* bytes, irrespective of their current positions in the
 36236 array) are passed more than once to the comparison function, the results shall be consistent with
 36237 one another. That is, they shall define a total ordering on the array.

36238 The contents of the array shall be sorted in ascending order according to a comparison function.
 36239 The *compar* argument is a pointer to the comparison function, which is called with two
 36240 arguments that point to the elements being compared. The application shall ensure that the
 36241 function returns an integer less than, equal to, or greater than 0, if the first argument is
 36242 considered respectively less than, equal to, or greater than the second. If two members compare
 36243 as equal, their order in the sorted array is unspecified.

36244 **RETURN VALUE**36245 The *qsort()* function shall not return a value.36246 **ERRORS**

36247 No errors are defined.

36248 **EXAMPLES**

36249 None.

36250 **APPLICATION USAGE**

36251 The comparison function need not compare every byte, so arbitrary data may be contained in
 36252 the elements in addition to the values being compared.

36253 **RATIONALE**

36254 The requirement that each argument (hereafter referred to as *p*) to the comparison function is a
 36255 pointer to elements of the array implies that for every call, for each argument separately, all of
 36256 the following expressions are nonzero:

```
36257       ((char *)p - (char *)base) % width == 0
36258       (char *)p >= (char *)base
36259       (char *)p < (char *)base + nel * width
```

36260 **FUTURE DIRECTIONS**

36261 None.

36262 **SEE ALSO**

36263 The Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

36264 **CHANGE HISTORY**

36265 First released in Issue 1. Derived from Issue 1 of the SVID.

36266 **Issue 6**

36267 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

36268 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 49 is applied, adding the last sentence to
36269 the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The
36270 RATIONALE is also updated. These changes are for alignment with the ISO C standard.

36271 **NAME**

36272 raise — send a signal to the executing process

36273 **SYNOPSIS**

36274 #include <signal.h>

36275 int raise(int *sig*);36276 **DESCRIPTION**

36277 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 36278 conflict between the requirements described here and the ISO C standard is unintentional. This
 36279 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36280 CX The *raise()* function shall send the signal *sig* to the executing thread or process. If a signal
 36281 handler is called, the *raise()* function shall not return until after the signal handler does.

36282 THR If the implementation supports the Threads option, the effect of the *raise()* function shall be
 36283 equivalent to calling:

36284 pthread_kill(pthread_self(), sig);

36285

36286 CX Otherwise, the effect of the *raise()* function shall be equivalent to calling:

36287 kill(getpid(), sig);

36288

36289 **RETURN VALUE**

36290 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned
 36291 and *errno* shall be set to indicate the error.

36292 **ERRORS**36293 The *raise()* function shall fail if:36294 CX [EINVAL] The value of the *sig* argument is an invalid signal number.36295 **EXAMPLES**

36296 None.

36297 **APPLICATION USAGE**

36298 None.

36299 **RATIONALE**

36300 The term “thread” is an extension to the ISO C standard.

36301 **FUTURE DIRECTIONS**

36302 None.

36303 **SEE ALSO**

36304 *kill()*, *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>,
 36305 <sys/types.h>

36306 **CHANGE HISTORY**

36307 First released in Issue 4. Derived from the ANSI C standard.

36308 **Issue 5**

36309 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

36310 **Issue 6**

36311 Extensions beyond the ISO C standard are marked.

36312 The following new requirements on POSIX implementations derive from alignment with the
36313 Single UNIX Specification:

- 36314 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 36315 • The [EINVAL] error condition is added.

36316 **NAME**

36317 rand, rand_r, srand — pseudo-random number generator

36318 **SYNOPSIS**

36319 #include <stdlib.h>

36320 int rand(void);

36321 TSF int rand_r(unsigned *seed);

36322 void srand(unsigned seed);

36323 **DESCRIPTION**

36324 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 36325 conflict between the requirements described here and the ISO C standard is unintentional. This
 36326 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36327 The *rand()* function shall compute a sequence of pseudo-random integers in the range
 36328 XSI [0,{RAND_MAX}] with a period of at least 2^{32} .

36329 CX The *rand()* function need not be reentrant. A function that is not required to be reentrant is not
 36330 required to be thread-safe.

36331 TSF The *rand_r()* function shall compute a sequence of pseudo-random integers in the range
 36332 [0,{RAND_MAX}]. (The value of the {RAND_MAX} macro shall be at least 32 767.)

36333 If *rand_r()* is called with the same initial value for the object pointed to by *seed* and that object is
 36334 not modified between successive returns and calls to *rand_r()*, the same sequence shall be
 36335 generated.

36336 The *srand()* function uses the argument as a seed for a new sequence of pseudo-random
 36337 numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same
 36338 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before
 36339 any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first
 36340 called with a seed value of 1.

36341 The implementation shall behave as if no function defined in this volume of
 36342 IEEE Std 1003.1-2001 calls *rand()* or *srand()*.

36343 **RETURN VALUE**36344 The *rand()* function shall return the next pseudo-random number in the sequence.36345 TSF The *rand_r()* function shall return a pseudo-random integer.36346 The *srand()* function shall not return a value.36347 **ERRORS**

36348 No errors are defined.

36349 **EXAMPLES**36350 **Generating a Pseudo-Random Number Sequence**

36351 The following example demonstrates how to generate a sequence of pseudo-random numbers.

36352 #include <stdio.h>

36353 #include <stdlib.h>

36354 ...

36355 long count, i;

36356 char *keyst;

36357 int elementlen, len;

36358 char c;

```

36359     ...
36360     /* Initial random number generator. */
36361     srand(1);

36362     /* Create keys using only lowercase characters */
36363     len = 0;
36364     for (i=0; i<count; i++) {
36365         while (len < elementlen) {
36366             c = (char) (rand() % 128);
36367             if (islower(c))
36368                 keystr[len++] = c;
36369         }

36370         keystr[len] = '\0';
36371         printf("%s Element%0*ld\n", keystr, elementlen, i);
36372         len = 0;
36373     }

```

Generating the Same Sequence on Different Machines

The following code defines a pair of functions that could be incorporated into applications wishing to ensure that the same sequence of numbers is generated across different machines.

```

36377 static unsigned long next = 1;
36378 int myrand(void) /* RAND_MAX assumed to be 32767. */
36379 {
36380     next = next * 1103515245 + 12345;
36381     return((unsigned)(next/65536) % 32768);
36382 }

36383 void mysrand(unsigned seed)
36384 {
36385     next = seed;
36386 }

```

APPLICATION USAGE

The *drand48()* function provides a much more elaborate random number generator.

The limitations on the amount of state that can be carried between one function call and another mean the *rand_r()* function can never be implemented in a way which satisfies all of the requirements on a pseudo-random number generator. Therefore this function should be avoided whenever non-trivial requirements (including safety) have to be fulfilled.

RATIONALE

The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams shared by all threads. Those two functions need not change, but there has to be mutual-exclusion that prevents interference between two threads concurrently accessing the random number generator.

With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded program:

1. A single per-process sequence of pseudo-random numbers that is shared by all threads that call *rand()*
2. A different sequence of pseudo-random numbers for each thread that calls *rand()*

36403 This is provided by the modified thread-safe function based on whether the seed value is global
36404 to the entire process or local to each thread.

36405 This does not address the known deficiencies of the *rand()* function implementations, which
36406 have been approached by maintaining more state. In effect, this specifies new thread-safe forms
36407 of a deficient function.

36408 **FUTURE DIRECTIONS**

36409 None.

36410 **SEE ALSO**

36411 *drand48()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

36412 **CHANGE HISTORY**

36413 First released in Issue 1. Derived from Issue 1 of the SVID.

36414 **Issue 5**

36415 The *rand_r()* function is included for alignment with the POSIX Threads Extension.

36416 A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.

36417 **Issue 6**

36418 Extensions beyond the ISO C standard are marked.

36419 The *rand_r()* function is marked as part of the Thread-Safe Functions option.

36420 **NAME**

36421 random — generate pseudo-random number

36422 **SYNOPSIS**

36423 xSI #include <stdlib.h>

36424 long random(void);

36425

36426 **DESCRIPTION**36427 Refer to *initstate()*.

36428 **NAME**

36429 pread, read — read from a file

36430 **SYNOPSIS**

36431 #include <unistd.h>

36432 xSI ssize_t pread(int *fildes*, void *buf, size_t *nbyte*, off_t *offset*);36433 ssize_t read(int *fildes*, void *buf, size_t *nbyte*);36434 **DESCRIPTION**

36435 The *read()* function shall attempt to read *nbyte* bytes from the file associated with the open file
 36436 descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on
 36437 the same pipe, FIFO, or terminal device is unspecified.

36438 Before any action described below is taken, and if *nbyte* is zero, the *read()* function may detect
 36439 and return errors as described below. In the absence of errors, or if error detection is not
 36440 performed, the *read()* function shall return zero and have no other results.

36441 On files that support seeking (for example, a regular file), the *read()* shall start at a position in
 36442 the file given by the file offset associated with *fildes*. The file offset shall be incremented by the
 36443 number of bytes actually read.

36444 Files that do not support seeking—for example, terminals—always read from the current
 36445 position. The value of a file offset associated with such a file is undefined.

36446 No data transfer shall occur past the current end-of-file. If the starting position is at or after the
 36447 end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent
 36448 *read()* requests is implementation-defined.

36449 If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

36450 When attempting to read from an empty pipe or FIFO:

- 36451 • If no process has the pipe open for writing, *read()* shall return 0 to indicate end-of-file.
- 36452 • If some process has the pipe open for writing and O_NONBLOCK is set, *read()* shall return
 36453 –1 and set *errno* to [EAGAIN].
- 36454 • If some process has the pipe open for writing and O_NONBLOCK is clear, *read()* shall block
 36455 the calling thread until some data is written or the pipe is closed by all processes that had the
 36456 pipe open for writing.

36457 When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and
 36458 has no data currently available:

- 36459 • If O_NONBLOCK is set, *read()* shall return –1 and set *errno* to [EAGAIN].
- 36460 • If O_NONBLOCK is clear, *read()* shall block the calling thread until some data becomes
 36461 available.
- 36462 • The use of the O_NONBLOCK flag has no effect if there is some data available.

36463 The *read()* function reads data previously written to a file. If any portion of a regular file prior to
 36464 the end-of-file has not been written, *read()* shall return bytes with value 0. For example, *lseek()*
 36465 allows the file offset to be set beyond the end of existing data in the file. If data is later written at
 36466 this point, subsequent reads in the gap between the previous end of data and the newly written
 36467 data shall return bytes with value 0 until data is written into the gap.

36468 Upon successful completion, where *nbyte* is greater than 0, *read()* shall mark for update the
 36469 *st_atime* field of the file, and shall return the number of bytes read. This number shall never be
 36470 greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the

36471	file is less than <i>nbyte</i> , if the <i>read()</i> request was interrupted by a signal, or if the file is a pipe or
36472	FIFO or special file and has fewer than <i>nbyte</i> bytes immediately available for reading. For
36473	example, a <i>read()</i> from a file associated with a terminal may return one typed line of data.
36474	If a <i>read()</i> is interrupted by a signal before it reads any data, it shall return <code>-1</code> with <i>errno</i> set to
36475	<code>[EINTR]</code> .
36476	If a <i>read()</i> is interrupted by a signal after it has successfully read some data, it shall return the
36477	number of bytes read.
36478	For regular files, no data transfer shall occur past the offset maximum established in the open
36479	file description associated with <i>fildev</i> .
36480	If <i>fildev</i> refers to a socket, <i>read()</i> shall be equivalent to <i>recv()</i> with no flags set.
36481 SIO	If the <code>O_DSYNC</code> and <code>O_RSYNC</code> bits have been set, read I/O operations on the file descriptor
36482	shall complete as defined by synchronized I/O data integrity completion. If the <code>O_SYNC</code> and
36483	<code>O_RSYNC</code> bits have been set, read I/O operations on the file descriptor shall complete as
36484	defined by synchronized I/O file integrity completion.
36485 SHM	If <i>fildev</i> refers to a shared memory object, the result of the <i>read()</i> function is unspecified.
36486 TYM	If <i>fildev</i> refers to a typed memory object, the result of the <i>read()</i> function is unspecified.
36487 XSR	A <i>read()</i> from a STREAMS file can read data in three different modes: <i>byte-stream</i> mode,
36488	<i>message-nondiscard</i> mode, and <i>message-discard</i> mode. The default shall be <i>byte-stream</i> mode. This
36489	can be changed using the <code>I_SRDOPT ioctl()</code> request, and can be tested with <code>I_GRDOPT ioctl()</code> .
36490	In <i>byte-stream</i> mode, <i>read()</i> shall retrieve data from the STREAM until as many bytes as were
36491	requested are transferred, or until there is no more data to be retrieved. <i>Byte-stream</i> mode
36492	ignores message boundaries.
36493	In STREAMS <i>message-nondiscard</i> mode, <i>read()</i> shall retrieve data until as many bytes as were
36494	requested are transferred, or until a message boundary is reached. If <i>read()</i> does not retrieve all
36495	the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by
36496	the next <i>read()</i> call. <i>Message-discard</i> mode also retrieves data until as many bytes as were
36497	requested are transferred, or a message boundary is reached. However, unread data remaining
36498	in a message after the <i>read()</i> returns shall be discarded, and shall not be available for a
36499	subsequent <i>read()</i> , <i>getmsg()</i> , or <i>getpmsg()</i> call.
36500	How <i>read()</i> handles zero-byte STREAMS messages is determined by the current read mode
36501	setting. In <i>byte-stream</i> mode, <i>read()</i> shall accept data until it has read <i>nbyte</i> bytes, or until there
36502	is no more data to read, or until a zero-byte message block is encountered. The <i>read()</i> function
36503	shall then return the number of bytes read, and place the zero-byte message back on the
36504	STREAM to be retrieved by the next <i>read()</i> , <i>getmsg()</i> , or <i>getpmsg()</i> . In <i>message-nondiscard</i> mode
36505	or <i>message-discard</i> mode, a zero-byte message shall return 0 and the message shall be removed
36506	from the STREAM. When a zero-byte message is read as the first message on a STREAM, the
36507	message shall be removed from the STREAM and 0 shall be returned, regardless of the read
36508	mode.
36509	A <i>read()</i> from a STREAMS file shall return the data in the message at the front of the STREAM
36510	head read queue, regardless of the priority band of the message.
36511	By default, STREAMs are in <i>control-normal</i> mode, in which a <i>read()</i> from a STREAMS file can
36512	only process messages that contain a data part but do not contain a control part. The <i>read()</i> shall
36513	fail if a message containing a control part is encountered at the STREAM head. This default
36514	action can be changed by placing the STREAM in either <i>control-data</i> mode or <i>control-discard</i>
36515	mode with the <code>I_SRDOPT ioctl()</code> command. In <i>control-data</i> mode, <i>read()</i> shall convert any
36516	control part to data and pass it to the application before passing any data part originally present

36517 in the same message. In control-discard mode, *read()* shall discard message control parts but
 36518 return to the process any data part in the message.

36519 In addition, *read()* shall fail if the STREAM head had processed an asynchronous error before the
 36520 call. In this case, the value of *errno* shall not reflect the result of *read()*, but reflect the prior error.
 36521 If a hangup occurs on the STREAM being read, *read()* shall continue to operate normally until
 36522 the STREAM head read queue is empty. Thereafter, it shall return 0.

36523 XSI The *pread()* function shall be equivalent to *read()*, except that it shall read from a given position
 36524 in the file without changing the file pointer. The first three arguments to *pread()* are the same as
 36525 *read()* with the addition of a fourth argument *offset* for the desired position inside the file. An
 36526 attempt to perform a *pread()* on a file that is incapable of seeking shall result in an error.

36527 RETURN VALUE

36528 XSI Upon successful completion, *read()* and *pread()* shall return a non-negative integer indicating the
 36529 number of bytes actually read. Otherwise, the functions shall return -1 and set *errno* to indicate
 36530 the error.

36531 ERRORS

36532 XSI The *read()* and *pread()* functions shall fail if:

36533 [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the process would be
 36534 delayed.

36535 [EBADF] The *fildev* argument is not a valid file descriptor open for reading.

36536 XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message
 36537 waiting to be read includes a control part.

36538 [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 36539 was transferred.

36540 XSR [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or
 36541 indirectly) downstream from a multiplexer.

36542 [EIO] The process is a member of a background process attempting to read from its
 36543 controlling terminal, the process is ignoring or blocking the SIGTTIN signal,
 36544 or the process group is orphaned. This error may also be generated for
 36545 implementation-defined reasons.

36546 XSI [EISDIR] The *fildev* argument refers to a directory and the implementation does not
 36547 allow the directory to be read using *read()* or *pread()*. The *readdir()* function
 36548 should be used instead.

36549 [EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before
 36550 the end-of-file, and the starting position is greater than or equal to the offset
 36551 maximum established in the open file description associated with *fildev*.

36552 The *read()* function shall fail if:

36553 [EAGAIN] or [EWOULDBLOCK]

36554 The file descriptor is for a socket, is marked O_NONBLOCK, and no data is
 36555 waiting to be received.

36556 [ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by
 36557 its peer.

36558 [ENOTCONN] A read was attempted on a socket that is not connected.

36559 [ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.

36560 XSI The `read()` and `pread()` functions may fail if:

36561 [EIO] A physical I/O error has occurred.

36562 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

36563 [ENOMEM] Insufficient memory was available to fulfill the request.

36564 [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.

36566 The `pread()` function shall fail, and the file pointer shall remain unchanged, if:

36567 XSI [EINVAL] The *offset* argument is invalid. The value is negative.

36568 XSI [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the file.

36570 XSI [ENXIO] A request was outside the capabilities of the device.

36571 XSI [ESPIPE] *fdes* is associated with a pipe or FIFO.

36572 EXAMPLES

36573 Reading Data into a Buffer

36574 The following example reads data from the file associated with the file descriptor *fd* into the
36575 buffer pointed to by *buf*.

```
36576 #include <sys/types.h>
36577 #include <unistd.h>
36578 ...
36579 char buf[20];
36580 size_t nbytes;
36581 ssize_t bytes_read;
36582 int fd;
36583 ...
36584 nbytes = sizeof(buf);
36585 bytes_read = read(fd, buf, nbytes);
36586 ...
```

36587 APPLICATION USAGE

36588 None.

36589 RATIONALE

36590 This volume of IEEE Std 1003.1-2001 does not specify the value of the file offset after an error is
36591 returned; there are too many cases. For programming errors, such as [EBADF], the concept is
36592 meaningless since no file is involved. For errors that are detected immediately, such as
36593 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,
36594 an updated value would be very useful and is the behavior of many implementations.

36595 Note that a `read()` of zero bytes does not modify *st_atime*. A `read()` that requests more than zero
36596 bytes, but returns zero, shall modify *st_atime*.

36597 Implementations are allowed, but not required, to perform error checking for `read()` requests of
36598 zero bytes.

Input and Output

The use of I/O with large byte counts has always presented problems. Ideas such as *lread()* and *lwrite()* (using and returning **longs**) were considered at one time. The current solution is to use abstract types on the ISO C standard function to *read()* and *write()*. The abstract types can be declared so that existing functions work, but can also be declared so that larger types can be represented in future implementations. It is presumed that whatever constraints limit the maximum range of **size_t** also limit portable I/O requests to the same range. This volume of IEEE Std 1003.1-2001 also limits the range further by requiring that the byte count be limited so that a signed return value remains meaningful. Since the return type is also a (signed) abstract type, the byte count can be defined by the implementation to be larger than an **int** can hold.

The standard developers considered adding atomicity requirements to a pipe or FIFO, but recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of reads of {PIPE_BUF} or any other size that would be an aid to applications portability.

This volume of IEEE Std 1003.1-2001 requires that no action be taken for *read()* or *write()* when *nbyte* is zero. This is not intended to take precedence over detection of errors (such as invalid buffer pointers or file descriptors). This is consistent with the rest of this volume of IEEE Std 1003.1-2001, but the phrasing here could be misread to require detection of the zero case before any other errors. A value of zero is to be considered a correct value, for which the semantics are a no-op.

I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the bytes from a single operation that started out together end up together, without interleaving from other I/O operations. It is a known attribute of terminals that this is not honored, and terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified. The behavior for other device types is also left unspecified, but the wording is intended to imply that future standards might choose to specify atomicity (or not).

There were recommendations to add format parameters to *read()* and *write()* in order to handle networked transfers among heterogeneous file system and base hardware types. Such a facility may be required for support by the OSI presentation of layer services. However, it was determined that this should correspond with similar C-language facilities, and that is beyond the scope of this volume of IEEE Std 1003.1-2001. The concept was suggested to the developers of the ISO C standard for their consideration as a possible area for future work.

In 4.3 BSD, a *read()* or *write()* that is interrupted by a signal before transferring any data does not by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition, there is an additional function, *select()*, whose purpose is to pause until specified activity (data to read, space to write, and so on) is detected on specified file descriptors. It is common in applications written for those systems for *select()* to be used before *read()* in situations (such as keyboard input) where interruption of I/O due to a signal is desired.

The issue of which files or file types are interruptible is considered an implementation design issue. This is often affected primarily by hardware and reliability issues.

There are no references to actions taken following an “unrecoverable error”. It is considered beyond the scope of this volume of IEEE Std 1003.1-2001 to describe what happens in the case of hardware errors.

Previous versions of IEEE Std 1003.1-2001 allowed two very different behaviors with regard to the handling of interrupts. In order to minimize the resulting confusion, it was decided that IEEE Std 1003.1-2001 should support only one of these behaviors. Historical practice on AT&T-derived systems was to have *read()* and *write()* return -1 and set *errno* to [EINTR] when interrupted after some, but not all, of the data requested had been transferred. However, the U.S. Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in

36647 which *read()* and *write()* return the number of bytes actually transferred before the interrupt. If
 36648 -1 is returned when any data is transferred, it is difficult to recover from the error on a seekable
 36649 device and impossible on a non-seekable device. Most new implementations support this
 36650 behavior. The behavior required by IEEE Std 1003.1-2001 is to return the number of bytes
 36651 transferred.

36652 IEEE Std 1003.1-2001 does not specify when an implementation that buffers *read()*s actually
 36653 moves the data into the user-supplied buffer, so an implementation may chose to do this at the
 36654 latest possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a
 36655 partial byte count, but rather to return -1 and set *errno* to [EINTR].

36656 Consideration was also given to combining the two previous options, and setting *errno* to
 36657 [EINTR] while returning a short count. However, not only is there no existing practice that
 36658 implements this, it is also contradictory to the idea that when *errno* is set, the function
 36659 responsible shall return -1.

36660 FUTURE DIRECTIONS

36661 None.

36662 SEE ALSO

36663 *fcntl()*, *ioctl()*, *lseek()*, *open()*, *pipe()*, *readv()*, the Base Definitions volume of
 36664 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, `<stropts.h>`, `<sys/uio.h>`,
 36665 `<unistd.h>`

36666 CHANGE HISTORY

36667 First released in Issue 1. Derived from Issue 1 of the SVID.

36668 Issue 5

36669 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 36670 Threads Extension.

36671 Large File Summit extensions are added.

36672 The *pread()* function is added.

36673 Issue 6

36674 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are
 36675 marked as part of the XSI STREAMS Option Group.

36676 The following new requirements on POSIX implementations derive from alignment with the
 36677 Single UNIX Specification:

36678 • The DESCRIPTION now states that if *read()* is interrupted by a signal after it has successfully
 36679 read some data, it returns the number of bytes read. In Issue 3, it was optional whether *read()*
 36680 returned the number of bytes read, or whether it returned -1 with *errno* set to [EINTR]. This
 36681 is a FIPS requirement.

36682 • In the DESCRIPTION, text is added to indicate that for regular files, no data transfer occurs
 36683 past the offset maximum established in the open file description associated with *files*. This
 36684 change is to support large files.

36685 • The [EOVERFLOW] mandatory error condition is added.

36686 • The [ENXIO] optional error condition is added.

36687 Text referring to sockets is added to the DESCRIPTION.

36688 The following changes were made to align with the IEEE P1003.1a draft standard:

36689 • The effect of reading zero bytes is clarified.

36690 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
36691 *read()* results are unspecified for typed memory objects.

36692 New RATIONALE is added to explain the atomicity requirements for input and output
36693 operations.

36694 The following error conditions are added for operations on sockets: [EAGAIN],
36695 [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].

36696 The [EIO] error is changed to “may fail”.

36697 The following error conditions are added for operations on sockets: [ENOBUFFS] and
36698 [ENOMEM].

36699 The *readv()* function is split out into a separate reference page.

36700 **NAME**

36701 readdir, readdir_r — read a directory

36702 **SYNOPSIS**

36703 #include <dirent.h>

36704 struct dirent *readdir(DIR *dirp);

36705 TSF int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,

36706 struct dirent **restrict result);

36707

36708 **DESCRIPTION**

36709 The type **DIR**, which is defined in the <**dirent.h**> header, represents a *directory stream*, which is
 36710 an ordered sequence of all the directory entries in a particular directory. Directory entries
 36711 represent files; files may be removed from a directory or added to a directory asynchronously to
 36712 the operation of *readdir()*.

36713 The *readdir()* function shall return a pointer to a structure representing the directory entry at the
 36714 current position in the directory stream specified by the argument *dirp*, and position the
 36715 directory stream at the next entry. It shall return a null pointer upon reaching the end of the
 36716 directory stream. The structure **dirent** defined in the <**dirent.h**> header describes a directory
 36717 entry.

36718 The *readdir()* function shall not return directory entries containing empty names. If entries for
 36719 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-
 36720 dot; otherwise, they shall not be returned.

36721 The pointer returned by *readdir()* points to data which may be overwritten by another call to
 36722 *readdir()* on the same directory stream. This data is not overwritten by another call to *readdir()*
 36723 on a different directory stream.

36724 If a file is removed from or added to the directory after the most recent call to *opendir()* or
 36725 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

36726 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()*
 36727 shall mark for update the *st_atime* field of the directory each time the directory is actually read.

36728 After a call to *fork()*, either the parent or child (but not both) may continue processing the
 36729 XSI directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes
 36730 use these functions, the result is undefined.

36731 If the entry names a symbolic link, the value of the *d_ino* member is unspecified.

36732 The *readdir()* function need not be reentrant. A function that is not required to be reentrant is not
 36733 required to be thread-safe.

36734 TSF The *readdir_r()* function shall initialize the **dirent** structure referenced by *entry* to represent the
 36735 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer
 36736 to this structure at the location referenced by *result*, and position the directory stream at the next
 36737 entry.

36738 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d_name*
 36739 members containing at least {NAME_MAX}+1 elements.

36740 Upon successful return, the pointer returned at **result* shall have the same value as the argument
 36741 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

36742 The *readdir_r()* function shall not return directory entries containing empty names.

36743 If a file is removed from or added to the directory after the most recent call to *opendir()* or
 36744 *rewinddir()*, whether a subsequent call to *readdir_r()* returns an entry for that file is unspecified.

36745 The *readdir_r()* function may buffer several directory entries per actual read operation; the
 36746 *readdir_r()* function shall mark for update the *st_atime* field of the directory each time the
 36747 directory is actually read.

36748 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If
 36749 *errno* is set to non-zero on return, an error occurred.

36750 RETURN VALUE

36751 Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**.
 36752 When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate
 36753 the error. When the end of the directory is encountered, a null pointer shall be returned and *errno*
 36754 is not changed.

36755 TSF If successful, the *readdir_r()* function shall return zero; otherwise, an error number shall be
 36756 returned to indicate the error.

36757 ERRORS

36758 The *readdir()* function shall fail if:

36759 [EOVERFLOW] One of the values in the structure to be returned cannot be represented
 36760 correctly.

36761 The *readdir_r()* function may fail if:

36762 [EBADF] The *dirp* argument does not refer to an open directory stream.

36763 [ENOENT] The current position of the directory stream is invalid.

36764 The *readdir_r()* function may fail if:

36765 [EBADF] The *dirp* argument does not refer to an open directory stream.

36766 EXAMPLES

36767 The following sample program searches the current directory for each of the arguments supplied
 36768 on the command line.

```
36769 #include <dirent.h>
36770 #include <errno.h>
36771 #include <stdio.h>
36772 #include <string.h>

36773 static void lookup(const char *arg)
36774 {
36775     DIR *dirp;
36776     struct dirent *dp;

36777     if ((dirp = opendir(".")) == NULL) {
36778         perror("couldn't open '.');
36779         return;
36780     }

36781     do {
36782         errno = 0;
36783         if ((dp = readdir(dirp)) != NULL) {
36784             if (strcmp(dp->d_name, arg) != 0)
36785                 continue;
```

```

36786         (void) printf("found %s\n", arg);
36787         (void) closedir(dirp);
36788         return;
36789     }
36790     } while (dp != NULL);
36791     if (errno != 0)
36792         perror("error reading directory");
36793     else
36794         (void) printf("failed to find %s\n", arg);
36795     (void) closedir(dirp);
36796     return;
36797 }
36798 int main(int argc, char *argv[])
36799 {
36800     int i;
36801     for (i = 1; i < argc; i++)
36802         lookup(argv[i]);
36803     return (0);
36804 }

```

36805 APPLICATION USAGE

36806 The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to
 36807 examine the contents of the directory.

36808 The *readdir_r()* function is thread-safe and shall return values in a user-supplied buffer instead
 36809 of possibly using a static data area that may be overwritten by each call.

36810 RATIONALE

36811 The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be
 36812 inferred.

36813 Historical implementations of *readdir()* obtain multiple directory entries on a single read
 36814 operation, which permits subsequent *readdir()* operations to operate from the buffered
 36815 information. Any wording that required each successful *readdir()* operation to mark the
 36816 directory *st_atime* field for update would disallow such historical performance-oriented
 36817 implementations.

36818 Since *readdir()* returns NULL when it detects an error and when the end of the directory is
 36819 encountered, an application that needs to tell the difference must set *errno* to zero before the call
 36820 and check it if NULL is returned. Since the function must not change *errno* in the second case
 36821 and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL
 36822 indicates end-of-directory; otherwise, an error.

36823 Routines to deal with this problem more directly were proposed:

```

36824 int derror (dirp)
36825 DIR *dirp;
36826 void clearderr (dirp)
36827 DIR *dirp;

```

36828 The first would indicate whether an error had occurred, and the second would clear the error
 36829 indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()*
 36830 not change *errno* when end-of-directory is encountered.

36831 An error or signal indicating that a directory has changed while open was considered but
36832 rejected.

36833 The thread-safe version of the directory reading function returns values in a user-supplied buffer
36834 instead of possibly using a static data area that may be overwritten by each call. Either the
36835 {NAME_MAX} compile-time constant or the corresponding *pathconf()* option can be used to
36836 determine the maximum sizes of returned pathnames.

36837 FUTURE DIRECTIONS

36838 None.

36839 SEE ALSO

36840 *closedir()*, *lstat()*, *opendir()*, *rewinddir()*, *symlink()*, the Base Definitions volume of
36841 IEEE Std 1003.1-2001, **<dirent.h>**, **<sys/types.h>**

36842 CHANGE HISTORY

36843 First released in Issue 2.

36844 Issue 5

36845 Large File Summit extensions are added.

36846 The *readdir_r()* function is included for alignment with the POSIX Threads Extension.

36847 A note indicating that the *readdir()* function need not be reentrant is added to the
36848 DESCRIPTION.

36849 Issue 6

36850 The *readdir_r()* function is marked as part of the Thread-Safe Functions option.

36851 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r()*.

36852 The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful
36853 return for the *readdir_r()* function.

36854 The following new requirements on POSIX implementations derive from alignment with the
36855 Single UNIX Specification:

- 36856 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
36857 required for conforming implementations of previous POSIX specifications, it was not
36858 required for UNIX applications.
- 36859 • A statement is added to the DESCRIPTION indicating the disposition of certain fields in
36860 **struct dirent** when an entry refers to a symbolic link.
- 36861 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
36862 files.
- 36863 • The [ENOENT] optional error condition is added.

36864 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
36865 its avoidance of possibly using a static data area.

36866 The **restrict** keyword is added to the *readdir_r()* prototype for alignment with the
36867 ISO/IEC 9899:1999 standard.

36868 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES
36869 section with a new example. |

36870 **NAME**

36871 readlink — read the contents of a symbolic link

36872 **SYNOPSIS**

36873 #include <unistd.h>

36874 ssize_t readlink(const char *restrict path, char *restrict buf,
 36875 size_t bufsize);

36876 **DESCRIPTION**

36877 The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the
 36878 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,
 36879 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to
 36880 contain the link content, the first *bufsize* bytes shall be placed in *buf*.

36881 If the value of *bufsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

36882 **RETURN VALUE**

36883 Upon successful completion, *readlink()* shall return the count of bytes placed in the buffer.
 36884 Otherwise, it shall return a value of -1, leave the buffer unchanged, and set *errno* to indicate the
 36885 error.

36886 **ERRORS**

36887 The *readlink()* function shall fail if:

36888 [EACCES] Search permission is denied for a component of the path prefix of *path*.

36889 [EINVAL] The *path* argument names a file that is not a symbolic link.

36890 [EIO] An I/O error occurred while reading from the file system.

36891 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 36892 argument.

36893 [ENAMETOOLONG]

36894 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 36895 component is longer than {NAME_MAX}.

36896 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

36897 [ENOTDIR] A component of the path prefix is not a directory.

36898 The *readlink()* function may fail if:

36899 [EACCES] Read permission is denied for the directory.

36900 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 36901 resolution of the *path* argument.

36902 [ENAMETOOLONG]

36903 As a result of encountering a symbolic link in resolution of the *path* argument,
 36904 the length of the substituted pathname string exceeded {PATH_MAX}.

36905 **EXAMPLES**36906 **Reading the Name of a Symbolic Link**

36907 The following example shows how to read the name of a symbolic link named **/modules/pass1**.

```
36908 #include <unistd.h>
36909 char buf[1024];
36910 ssize_t len;
36911 ...
36912 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
36913     buf[len] = '\0';
```

36914 **APPLICATION USAGE**

36915 Conforming applications should not assume that the returned contents of the symbolic link are
 36916 null-terminated.

36917 **RATIONALE**

36918 Since IEEE Std 1003.1-2001 does not require any association of file times with symbolic links,
 36919 there is no requirement that file times be updated by *readlink()*. The type associated with *bufsiz*
 36920 is a **size_t** in order to be consistent with both the ISO C standard and the definition of *read()*.
 36921 The behavior specified for *readlink()* when *bufsiz* is zero represents historical practice. For this
 36922 case, the standard developers considered a change whereby *readlink()* would return the number
 36923 of non-null bytes contained in the symbolic link with the buffer *buf* remaining unchanged;
 36924 however, since the **stat** structure member *st_size* value can be used to determine the size of
 36925 buffer necessary to contain the contents of the symbolic link as returned by *readlink()*, this
 36926 proposal was rejected, and the historical practice retained.

36927 **FUTURE DIRECTIONS**

36928 None.

36929 **SEE ALSO**

36930 *lstat()*, *stat()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

36931 **CHANGE HISTORY**

36932 First released in Issue 4, Version 2.

36933 **Issue 5**

36934 Moved from X/OPEN UNIX extension to BASE.

36935 **Issue 6**

36936 The return type is changed to **ssize_t**, to align with the IEEE P1003.1a draft standard.

36937 The following new requirements on POSIX implementations derive from alignment with the
 36938 Single UNIX Specification:

- 36939 • This function is made mandatory.
- 36940 • In this function it is possible for the return value to exceed the range of the type **ssize_t** (since
 36941 **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of
 36942 the **size_t** object is added to the description to resolve this conflict.

36943 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 36944 • The FUTURE DIRECTIONS section is changed to None.

36945 The following changes were made to align with the IEEE P1003.1a draft standard:

- 36946 • The [ELOOP] optional error condition is added.

36947 The **restrict** keyword is added to the *readlink()* prototype for alignment with the
36948 ISO / IEC 9899:1999 standard.

36949 **NAME**

36950 readv — read a vector

36951 **SYNOPSIS**

36952 xSI #include <sys/uio.h>

36953 ssize_t readv(int *fildes*, const struct iovec **iov*, int *iovcnt*);

36954

36955 **DESCRIPTION**

36956 The *readv()* function shall be equivalent to *read()*, except as described below. The *readv()*
 36957 function shall place the input data into the *iovcnt* buffers specified by the members of the *iov*
 36958 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*−1]. The *iovcnt* argument is valid if greater than 0 and less than
 36959 or equal to {IOV_MAX}.

36960 Each *iovec* entry specifies the base address and length of an area in memory where data should
 36961 be placed. The *readv()* function shall always fill an area completely before proceeding to the
 36962 next.

36963 Upon successful completion, *readv()* shall mark for update the *st_atime* field of the file.

36964 **RETURN VALUE**36965 Refer to *read()*.36966 **ERRORS**36967 Refer to *read()*.36968 In addition, the *readv()* function shall fail if:

36969 [EINVAL] The sum of the *iov_len* values in the *iov* array overflowed an **ssize_t**.

36970 The *readv()* function may fail if:

36971 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

36972 **EXAMPLES**36973 **Reading Data into an Array**

36974 The following example reads data from the file associated with the file descriptor *fd* into the
 36975 buffers specified by members of the *iov* array.

```

36976 #include <sys/types.h>
36977 #include <sys/uio.h>
36978 #include <unistd.h>
36979 ...
36980 ssize_t bytes_read;
36981 int fd;
36982 char buf0[20];
36983 char buf1[30];
36984 char buf2[40];
36985 int iovcnt;
36986 struct iovec iov[3];

36987 iov[0].iov_base = buf0;
36988 iov[0].iov_len = sizeof(buf0);
36989 iov[1].iov_base = buf1;
36990 iov[1].iov_len = sizeof(buf1);
36991 iov[2].iov_base = buf2;
36992 iov[2].iov_len = sizeof(buf2);

```



```
36993     ...
36994     iovcnt = sizeof(iov) / sizeof(struct iovec);
36995     bytes_read = readv(fd, iov, iovcnt);
36996     ...
```

36997 **APPLICATION USAGE**

36998 None.

36999 **RATIONALE**

37000 Refer to *read()*.

37001 **FUTURE DIRECTIONS**

37002 None.

37003 **SEE ALSO**

37004 *read()*, *writew()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/uio.h**>

37005 **CHANGE HISTORY**

37006 First released in Issue 4, Version 2.

37007 **Issue 6**

37008 Split out from the *read()* reference page.

37009 **NAME**

37010 realloc — memory reallocator

37011 **SYNOPSIS**

37012 #include <stdlib.h>

37013 void *realloc(void *ptr, size_t size);

37014 **DESCRIPTION**

37015 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37016 conflict between the requirements described here and the ISO C standard is unintentional. This
 37017 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37018 The *realloc()* function shall change the size of the memory object pointed to by *ptr* to the size
 37019 specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new
 37020 and old sizes. If the new size of the memory object would require movement of the object, the
 37021 space for the previous instantiation of the object is freed. If the new size is larger, the contents of
 37022 the newly allocated portion of the object are unspecified. If *size* is 0 and *ptr* is not a null pointer,
 37023 the object pointed to is freed. If the space cannot be allocated, the object shall remain unchanged.

37024 If *ptr* is a null pointer, *realloc()* shall be equivalent to *malloc()* for the specified size.

37025 If *ptr* does not match a pointer returned earlier by *calloc()*, *malloc()*, or *realloc()* or if the space has
 37026 previously been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

37027 The order and contiguity of storage allocated by successive calls to *realloc()* is unspecified. The
 37028 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 37029 a pointer to any type of object and then used to access such an object in the space allocated (until
 37030 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
 37031 disjoint from any other object. The pointer returned shall point to the start (lowest byte address)
 37032 of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

37033 **RETURN VALUE**

37034 Upon successful completion with a size not equal to 0, *realloc()* shall return a pointer to the
 37035 (possibly moved) allocated space. If *size* is 0, either a null pointer or a unique pointer that can be
 37036 successfully passed to *free()* shall be returned. If there is not enough available memory, *realloc()*
 37037 CX shall return a null pointer and set *errno* to [ENOMEM].

37038 **ERRORS**37039 The *realloc()* function shall fail if:

37040 CX [ENOMEM] Insufficient memory is available.

37041 **EXAMPLES**

37042 None.

37043 **APPLICATION USAGE**

37044 None.

37045 **RATIONALE**

37046 None.

37047 **FUTURE DIRECTIONS**

37048 None.

37049 **SEE ALSO**37050 *calloc()*, *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

37051 **CHANGE HISTORY**

37052 First released in Issue 1. Derived from Issue 1 of the SVID.

37053 **Issue 6**

37054 Extensions beyond the ISO C standard are marked.

37055 The following new requirements on POSIX implementations derive from alignment with the
37056 Single UNIX Specification:

- 37057 • In the RETURN VALUE section, if there is not enough available memory, the setting of *errno*
37058 to [ENOMEM] is added.
- 37059 • The [ENOMEM] error condition is added.

37060 **NAME**

37061 realpath — resolve a pathname

37062 **SYNOPSIS**

37063 xSI #include <stdlib.h>

```
37064       char *realpath(const char *restrict file_name,
37065                      char *restrict resolved_name);
```

37066

37067 **DESCRIPTION**

37068 The *realpath()* function shall derive, from the pathname pointed to by *file_name*, an absolute
 37069 pathname that names the same file, whose resolution does not involve '.', '..', or symbolic
 37070 links. The generated pathname shall be stored as a null-terminated string, up to a maximum of
 37071 {PATH_MAX} bytes, in the buffer pointed to by *resolved_name*.

37072 If *resolved_name* is a null pointer, the behavior of *realpath()* is implementation-defined.

37073 **RETURN VALUE**

37074 Upon successful completion, *realpath()* shall return a pointer to the resolved name. Otherwise,
 37075 *realpath()* shall return a null pointer and set *errno* to indicate the error, and the contents of the
 37076 buffer pointed to by *resolved_name* are undefined.

37077 **ERRORS**

37078 The *realpath()* function shall fail if:

37079	[EACCES]	Read or search permission was denied for a component of <i>file_name</i> .
37080	[EINVAL]	The <i>file_name</i> argument is a null pointer.
37081	[EIO]	An error occurred while reading from the file system.
37082	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
37083		
37084	[ENAMETOOLONG]	
37085		The length of the <i>file_name</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
37086		
37087	[ENOENT]	A component of <i>file_name</i> does not name an existing file or <i>file_name</i> points to an empty string.
37088		
37089	[ENOTDIR]	A component of the path prefix is not a directory.

37090 The *realpath()* function may fail if:

37091	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
37092		
37093	[ENAMETOOLONG]	
37094		Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
37095		
37096	[ENOMEM]	Insufficient storage space is available.

37097 **EXAMPLES**37098 **Generating an Absolute Pathname**

37099 The following example generates an absolute pathname for the file identified by the *symlinkpath*
 37100 argument. The generated pathname is stored in the *actualpath* array.

```
37101 #include <stdlib.h>
37102 ...
37103 char *symlinkpath = "/tmp/symlink/file";
37104 char actualpath [PATH_MAX+1];
37105 char *ptr;
37106 ptr = realpath(symlinkpath, actualpath);
```

37107 **APPLICATION USAGE**

37108 None.

37109 **RATIONALE**

37110 Since the maximum pathname length is arbitrary unless {PATH_MAX} is defined, an application
 37111 generally cannot supply a *resolved_name* buffer with size {{PATH_MAX}+1}.

37112 **FUTURE DIRECTIONS**

37113 In the future, passing a null pointer to *realpath()* for the *resolved_name* argument may be defined
 37114 to have *realpath()* allocate space for the generated pathname.

37115 **SEE ALSO**

37116 *getcwd()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

37117 **CHANGE HISTORY**

37118 First released in Issue 4, Version 2.

37119 **Issue 5**

37120 Moved from X/OPEN UNIX extension to BASE.

37121 **Issue 6**

37122 The **restrict** keyword is added to the *realpath()* prototype for alignment with the
 37123 ISO/IEC 9899:1999 standard.

37124 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 37125 [ELOOP] error condition is added.

37126 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the
 37127 DESCRIPTION for the case when *resolved_name* is a null pointer, changing the [EINVAL] error
 37128 text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.

37129 **NAME**

37130 recv — receive a message from a connected socket

37131 **SYNOPSIS**

37132 #include <sys/socket.h>

37133 ssize_t recv(int *socket*, void **buffer*, size_t *length*, int *flags*);37134 **DESCRIPTION**

37135 The *recv()* function shall receive a message from a connection-mode or connectionless-mode
 37136 socket. It is normally used with connected sockets because it does not permit the application to
 37137 retrieve the source address of received data.

37138 The *recv()* function takes the following arguments:

37139	<i>socket</i>	Specifies the socket file descriptor.
37140	<i>buffer</i>	Points to a buffer where the message should be stored.
37141	<i>length</i>	Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.
37142	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
37144	MSG_PEEK	Peeks at an incoming message. The data is treated as unread and the next <i>recv()</i> or similar function shall still return this data.
37146	MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
37148	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

37154 The *recv()* function shall return the length of the message written to the buffer pointed to by the
 37155 *buffer* argument. For message-based sockets, such as SOCK_DGRAM and SOCK_SEQPACKET,
 37156 the entire message shall be read in a single operation. If a message is too long to fit in the
 37157 supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be
 37158 discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be
 37159 ignored. In this case, data shall be returned to the user as soon as it becomes available, and no
 37160 data shall be discarded.

37161 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 37162 message.

37163 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 37164 descriptor, *recv()* shall block until a message arrives. If no messages are available at the socket
 37165 and O_NONBLOCK is set on the socket's file descriptor, *recv()* shall fail and set *errno* to
 37166 [EAGAIN] or [EWOULDBLOCK].

37167 **RETURN VALUE**

37168 Upon successful completion, *recv()* shall return the length of the message in bytes. If no
 37169 messages are available to be received and the peer has performed an orderly shutdown, *recv()*
 37170 shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

37171 **ERRORS**37172 The *recv()* function shall fail if:

37173 [EAGAIN] or [EWOULDBLOCK]

37174 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
 37175 to be received; or MSG_OOB is set and no out-of-band data is available and
 37176 either the socket's file descriptor is marked O_NONBLOCK or the socket does
 37177 not support blocking to await out-of-band data.

37178 [EBADF] The *socket* argument is not a valid file descriptor.

37179 [ECONNRESET] A connection was forcibly closed by a peer.

37180 [EINTR] The *recv()* function was interrupted by a signal that was caught, before any
 37181 data was available.

37182 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

37183 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

37184 [ENOTSOCK] The *socket* argument does not refer to a socket.

37185 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.

37186 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
 37187 transmission timeout on active connection.

37188 The *recv()* function may fail if:

37189 [EIO] An I/O error occurred while reading from or writing to the file system.

37190 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

37191 [ENOMEM] Insufficient memory was available to fulfill the request.

37192 **EXAMPLES**

37193 None.

37194 **APPLICATION USAGE**

37195 The *recv()* function is equivalent to *recvfrom()* with a zero *address_len* argument, and to *read()* if
 37196 no flags are used.

37197 The *select()* and *poll()* functions can be used to determine when data is available to be received.37198 **RATIONALE**

37199 None.

37200 **FUTURE DIRECTIONS**

37201 None.

37202 **SEE ALSO**

37203 *poll()*, *read()*, *recvmsg()*, *recvfrom()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*,
 37204 *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>

37205 **CHANGE HISTORY**

37206 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37207 **NAME**

37208 recvfrom — receive a message from a socket

37209 **SYNOPSIS**

37210 #include <sys/socket.h>

```
37211       ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
37212                       int flags, struct sockaddr *restrict address,
37213                       socklen_t *restrict address_len);
```

37214 **DESCRIPTION**

37215 The *recvfrom()* function shall receive a message from a connection-mode or connectionless-mode
 37216 socket. It is normally used with connectionless-mode sockets because it permits the application
 37217 to retrieve the source address of received data.

37218 The *recvfrom()* function takes the following arguments:

37219	<i>socket</i>	Specifies the socket file descriptor.
37220	<i>buffer</i>	Points to the buffer where the message should be stored.
37221	<i>length</i>	Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.
37222	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed 37223 by logically OR'ing zero or more of the following values:
37224	MSG_PEEK	Peeks at an incoming message. The data is treated as unread 37225 and the next <i>recvfrom()</i> or similar function shall still return 37226 this data.
37227	MSG_OOB	Requests out-of-band data. The significance and semantics 37228 of out-of-band data are protocol-specific.
37229	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function 37230 block until the full amount of data can be returned. The 37231 function may return the smaller amount of data if the socket 37232 is a message-based socket, if a signal is caught, if the 37233 connection is terminated, if MSG_PEEK was specified, or if 37234 an error is pending for the socket.
37235	<i>address</i>	A null pointer, or points to a sockaddr structure in which the sending address 37236 is to be stored. The length and format of the address depend on the address 37237 family of the socket.
37238	<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> 37239 argument.

37240 The *recvfrom()* function shall return the length of the message written to the buffer pointed to by
 37241 RS the *buffer* argument. For message-based sockets, such as SOCK_RAW, SOCK_DGRAM, and
 37242 SOCK_SEQPACKET, the entire message shall be read in a single operation. If a message is too
 37243 long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess
 37244 bytes shall be discarded. For stream-based sockets, such as SOCK_STREAM, message
 37245 boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes
 37246 available, and no data shall be discarded.

37247 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 37248 message.

37249 Not all protocols provide the source address for messages. If the *address* argument is not a null
 37250 pointer and the protocol provides the source address of messages, the source address of the

37251 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument,
 37252 and the length of this address shall be stored in the object pointed to by the *address_len*
 37253 argument.

37254 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
 37255 the stored address shall be truncated.

37256 If the *address* argument is not a null pointer and the protocol does not provide the source address
 37257 of messages, the value stored in the object pointed to by *address* is unspecified.

37258 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 37259 descriptor, *recvfrom()* shall block until a message arrives. If no messages are available at the
 37260 socket and O_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno*
 37261 to [EAGAIN] or [EWOULDBLOCK].

37262 RETURN VALUE

37263 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no
 37264 messages are available to be received and the peer has performed an orderly shutdown,
 37265 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the
 37266 error.

37267 ERRORS

37268 The *recvfrom()* function shall fail if:

37269 [EAGAIN] or [EWOULDBLOCK]

37270 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
 37271 to be received; or MSG_OOB is set and no out-of-band data is available and
 37272 either the socket's file descriptor is marked O_NONBLOCK or the socket does
 37273 not support blocking to await out-of-band data.

37274 [EBADF] The *socket* argument is not a valid file descriptor.

37275 [ECONNRESET] A connection was forcibly closed by a peer.

37276 [EINTR] A signal interrupted *recvfrom()* before any data was available.

37277 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

37278 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

37279 [ENOTSOCK] The *socket* argument does not refer to a socket.

37280 [EOPNOTSUPP] The specified flags are not supported for this socket type.

37281 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
 37282 transmission timeout on active connection.

37283 The *recvfrom()* function may fail if:

37284 [EIO] An I/O error occurred while reading from or writing to the file system.

37285 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

37286 [ENOMEM] Insufficient memory was available to fulfill the request.

37287 **EXAMPLES**

37288 None.

37289 **APPLICATION USAGE**37290 The *select()* and *poll()* functions can be used to determine when data is available to be received.37291 **RATIONALE**

37292 None.

37293 **FUTURE DIRECTIONS**

37294 None.

37295 **SEE ALSO**

37296 *poll()*, *read()*, *recv()*, *recvmsg()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, *write()*,
37297 the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>

37298 **CHANGE HISTORY**

37299 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37300 **NAME**

37301 recvmsg — receive a message from a socket

37302 **SYNOPSIS**

37303 #include <sys/socket.h>

37304 ssize_t recvmsg(int *socket*, struct msghdr **message*, int *flags*);37305 **DESCRIPTION**

37306 The *recvmsg()* function shall receive a message from a connection-mode or connectionless-mode
 37307 socket. It is normally used with connectionless-mode sockets because it permits the application
 37308 to retrieve the source address of received data.

37309 The *recvmsg()* function takes the following arguments:

37310	<i>socket</i>	Specifies the socket file descriptor.
37311	<i>message</i>	Points to a msghdr structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored on input, but may contain meaningful values on output.
37312		
37313		
37314		
37315	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
37316		
37317	MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
37318		
37319	MSG_PEEK	Peeks at the incoming message.
37320	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.
37321		
37322		
37323		
37324		
37325		

37326 The *recvmsg()* function shall receive messages from unconnected or connected sockets and shall
 37327 return the length of the message.

37328 The *recvmsg()* function shall return the total length of the message. For message-based sockets,
 37329 such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message shall be read in a single
 37330 operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is not set in the
 37331 *flags* argument, the excess bytes shall be discarded, and MSG_TRUNC shall be set in the
 37332 *msg_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK_STREAM,
 37333 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it
 37334 becomes available, and no data shall be discarded.

37335 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 37336 message.

37337 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 37338 descriptor, *recvmsg()* shall block until a message arrives. If no messages are available at the
 37339 socket and O_NONBLOCK is set on the socket's file descriptor, the *recvmsg()* function shall fail
 37340 and set *errno* to [EAGAIN] or [EWOULDBLOCK].

37341 In the **msghdr** structure, the *msg_name* and *msg_namelen* members specify the source address if
 37342 the socket is unconnected. If the socket is connected, the *msg_name* and *msg_namelen* members
 37343 shall be ignored. The *msg_name* member may be a null pointer if no names are desired or
 37344 required. The *msg_iov* and *msg_iovlen* fields are used to specify where the received data shall be

37345 stored. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of
 37346 this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field
 37347 gives its size in bytes. Each storage area indicated by *msg_iov* is filled with received data in turn
 37348 until all of the received data is stored or all of the areas have been filled.

37349 Upon successful completion, the *msg_flags* member of the message header shall be the bitwise-
 37350 inclusive OR of all of the following flags that indicate conditions detected for the received
 37351 message:

37352	MSG_EOR	End-of-record was received (if supported by the protocol).
37353	MSG_OOB	Out-of-band data was received.
37354	MSG_TRUNC	Normal data was truncated.
37355	MSG_CTRUNC	Control data was truncated.

37356 RETURN VALUE

37357 Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no
 37358 messages are available to be received and the peer has performed an orderly shutdown,
 37359 *recvmsg()* shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

37360 ERRORS

37361 The *recvmsg()* function shall fail if:

37362	[EAGAIN] or [EWOULDBLOCK]	
37363		The socket's file descriptor is marked O_NONBLOCK and no data is waiting
37364		to be received; or MSG_OOB is set and no out-of-band data is available and
37365		either the socket's file descriptor is marked O_NONBLOCK or the socket does
37366		not support blocking to await out-of-band data.

37367	[EBADF]	The <i>socket</i> argument is not a valid open file descriptor.

37368	[ECONNRESET]	A connection was forcibly closed by a peer.

37369	[EINTR]	This function was interrupted by a signal before any data was available.

37370	[EINVAL]	The sum of the <i>iov_len</i> values overflows a ssize_t , or the MSG_OOB flag is set
37371		and no out-of-band data is available.

37372	[EMSGSIZE]	The <i>msg_iovlen</i> member of the msghdr structure pointed to by <i>message</i> is less
37373		than or equal to 0, or is greater than {IOV_MAX}.

37374	[ENOTCONN]	A receive is attempted on a connection-mode socket that is not connected.

37375	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.

37376	[EOPNOTSUPP]	The specified flags are not supported for this socket type.

37377	[ETIMEDOUT]	The connection timed out during connection establishment, or due to a
37378		transmission timeout on active connection.

37379 The *recvmsg()* function may fail if:

37380	[EIO]	An I/O error occurred while reading from or writing to the file system.

37381	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.

37382	[ENOMEM]	Insufficient memory was available to fulfill the request.

37383 **EXAMPLES**

37384 None.

37385 **APPLICATION USAGE**37386 The *select()* and *poll()* functions can be used to determine when data is available to be received.37387 **RATIONALE**

37388 None.

37389 **FUTURE DIRECTIONS**

37390 None.

37391 **SEE ALSO**37392 *poll()*, *recv()*, *recvfrom()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, the Base37393 Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>37394 **CHANGE HISTORY**

37395 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37396 **NAME**

37397 regcomp, regerror, regex, regfree — regular expression matching

37398 **SYNOPSIS**

37399 #include <regex.h>

```

37400 int regcomp(regex_t *restrict preg, const char *restrict pattern,
37401             int cflags);
37402 size_t regerror(int errcode, const regex_t *restrict preg,
37403                char *restrict errbuf, size_t errbuf_size);
37404 int regexexec(const regex_t *restrict preg, const char *restrict string,
37405               size_t nmatch, regmatch_t pmatch[restrict], int eflags);
37406 void regfree(regex_t *preg);

```

37407 **DESCRIPTION**

37408 These functions interpret *basic* and *extended* regular expressions as described in the Base
 37409 Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular Expressions.

37410 The **regex_t** structure is defined in <**regex.h**> and contains at least the following member:

Member Type	Member Name	Description
size_t	re_nsub	Number of parenthesized subexpressions.

37414 The **regmatch_t** structure is defined in <**regex.h**> and contains at least the following members:

Member Type	Member Name	Description
regoff_t	rm_so	Byte offset from start of <i>string</i> to start of substring.
regoff_t	rm_eo	Byte offset from start of <i>string</i> of the first character after the end of substring.

37420 The *regcomp()* function shall compile the regular expression contained in the string pointed to by
 37421 the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags*
 37422 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in
 37423 the <**regex.h**> header:

37424	REG_EXTENDED	Use Extended Regular Expressions.
37425	REG_ICASE	Ignore case in match. (See the Base Definitions volume of 37426 IEEE Std 1003.1-2001, Chapter 9, Regular Expressions.)
37427	REG_NOSUB	Report only success / fail in <i>regexexec()</i> .
37428	REG_NEWLINE	Change the handling of <newline>s, as described in the text.

37429 The default regular expression type for *pattern* is a Basic Regular Expression. The application can
 37430 specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

37431 If the REG_NOSUB flag was not set in *cflags*, then *regcomp()* shall set *re_nsub* to the number of
 37432 parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "()" in
 37433 extended regular expressions) found in *pattern*.

37434 The *regexexec()* function compares the null-terminated string specified by *string* with the compiled
 37435 regular expression *preg* initialized by a previous call to *regcomp()*. If it finds a match, *regexexec()*
 37436 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The
 37437 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are
 37438 defined in the <**regex.h**> header:

37439 REG_NOTBOL The first character of the string pointed to by *string* is not the beginning of the
 37440 line. Therefore, the circumflex character ('^'), when taken as a special
 37441 character, shall not match the beginning of *string*.

37442 REG_NOTEOL The last character of the string pointed to by *string* is not the end of the line.
 37443 Therefore, the dollar sign ('\$'), when taken as a special character, shall not
 37444 match the end of *string*.

37445 If *nmatch* is 0 or REG_NOSUB was set in the *cflags* argument to *regcomp()*, then *regexec()* shall
 37446 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument
 37447 points to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that
 37448 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions
 37449 of *pattern*: *pmatch[i].rm_so* shall be the byte offset of the beginning and *pmatch[i].rm_eo* shall be
 37450 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th
 37451 matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* identify the substring that
 37452 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]*
 37453 shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself
 37454 counts as a subexpression), then *regexec()* shall still do the match, but shall record only the first
 37455 *nmatch* substrings.

37456 When matching a basic or extended regular expression, any given parenthesized subexpression
 37457 of *pattern* might participate in the match of several different substrings of *string*, or it might not
 37458 match any substring even though the pattern as a whole did match. The following rules shall be
 37459 used to determine which substrings to report in *pmatch* when matching regular expressions:

- 37460 1. If subexpression *i* in a regular expression is not contained within another subexpression,
 37461 and it participated in the match several times, then the byte offsets in *pmatch[i]* shall
 37462 delimit the last such match.
- 37463 2. If subexpression *i* is not contained within another subexpression, and it did not participate
 37464 in an otherwise successful match, the byte offsets in *pmatch[i]* shall be -1. A subexpression
 37465 does not participate in the match when:

37466 ' * ' or "\{\}" appears immediately after the subexpression in a basic regular
 37467 expression, or ' * ', ' ? ', or "{ }" appears immediately after the subexpression in an
 37468 extended regular expression, and the subexpression did not match (matched 0 times)

37469 or:

37470 ' | ' is used in an extended regular expression to select this subexpression or another,
 37471 and the other subexpression matched.

- 37472 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained
 37473 within any other subexpression that is contained within *j*, and a match of subexpression *j*
 37474 is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in
 37475 *pmatch[i]* shall be as described in 1. and 2. above, but within the substring reported in
 37476 *pmatch[j]* rather than the whole string. The offsets in *pmatch[i]* are still relative to the start
 37477 of *string*.

- 37478 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are -1,
 37479 then the pointers in *pmatch[i]* shall also be -1.

- 37480 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be
 37481 the byte offset of the character or null terminator immediately following the zero-length
 37482 string.

37483 If, when *regexec()* is called, the locale is different from when the regular expression was
 37484 compiled, the result is undefined.

37485 If REG_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an
 37486 ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary
 37487 character except as follows:

- 37488 1. A <newline> in *string* shall not be matched by a period outside a bracket expression or by
 37489 any form of a non-matching list (see the Base Definitions volume of IEEE Std 1003.1-2001,
 37490 Chapter 9, Regular Expressions).
- 37491 2. A circumflex ('^') in *pattern*, when used to specify expression anchoring (see the Base
 37492 Definitions volume of IEEE Std 1003.1-2001, Section 9.3.8, BRE Expression Anchoring),
 37493 shall match the zero-length string immediately after a <newline> in *string*, regardless of
 37494 the setting of REG_NOTBOL.
- 37495 3. A dollar sign ('\$') in *pattern*, when used to specify expression anchoring, shall match the
 37496 zero-length string immediately before a <newline> in *string*, regardless of the setting of
 37497 REG_NOTEOL.

37498 The *regfree()* function frees any memory allocated by *regcomp()* associated with *preg*.

37499 The following constants are defined as error return values:

37500	REG_NOMATCH	<i>regexec()</i> failed to match.
37501	REG_BADPAT	Invalid regular expression.
37502	REG_ECOLLATE	Invalid collating element referenced.
37503	REG_ETYPE	Invalid character class type referenced.
37504	REG_EESCAPE	Trailing '\\' in pattern.
37505	REG_ESUBREG	Number in "\digit" invalid or in error.
37506	REG_EBRACK	"[]" imbalance.
37507	REG_EPAREN	"\(\)" or "()" imbalance.
37508	REG_EBRACE	"\{\}" imbalance.
37509	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than 37510 two numbers, first larger than second.
37511	REG_ERANGE	Invalid endpoint in range expression.
37512	REG_ESPACE	Out of memory.
37513	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.

37514 The *regerror()* function provides a mapping from error codes returned by *regcomp()* and
 37515 *regexec()* to unspecified printable strings. It generates a string corresponding to the value of the
 37516 *errcode* argument, which the application shall ensure is the last non-zero value returned by
 37517 *regcomp()* or *regexec()* with the given value of *preg*. If *errcode* is not such a value, the content of
 37518 the generated string is unspecified.

37519 If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec()* or *regcomp()*,
 37520 the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not
 37521 be as detailed under some implementations.

37522 If the *errbuf_size* argument is not 0, *regerror()* shall place the generated string into the buffer of
 37523 size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit
 37524 in the buffer, *regerror()* shall truncate the string and null-terminate the result.

37525 If *errbuf_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer
 37526 needed to hold the generated string.

37527 If the *preg* argument to *regex()* or *regfree()* is not a compiled regular expression returned by
 37528 *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression
 37529 after it is given to *regfree()*.

37530 RETURN VALUE

37531 Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an
 37532 integer value indicating an error as described in **<regex.h>**, and the content of *preg* is undefined.
 37533 If a code is returned, the interpretation shall be as given in **<regex.h>**.

37534 If *regcomp()* detects an invalid RE, it may return REG_BADPAT, or it may return one of the error
 37535 codes that more precisely describes the error.

37536 Upon successful completion, the *regex()* function shall return 0. Otherwise, it shall return
 37537 REG_NOMATCH to indicate no match.

37538 Upon successful completion, the *regerror()* function shall return the number of bytes needed to
 37539 hold the entire generated string, including the null termination. If the return value is greater than
 37540 *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

37541 The *regfree()* function shall not return a value.

37542 ERRORS

37543 No errors are defined.

37544 EXAMPLES

```
37545 #include <regex.h>
37546 /*
37547  * Match string against the extended regular expression in
37548  * pattern, treating errors as no match.
37549  *
37550  * Return 1 for match, 0 for no match.
37551  */
37552 int
37553 match(const char *string, char *pattern)
37554 {
37555     int    status;
37556     regex_t re;
37557     if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
37558         return(0);      /* Report error. */
37559     }
37560     status = regexexec(&re, string, (size_t) 0, NULL, 0);
37561     regfree(&re);
37562     if (status != 0) {
37563         return(0);      /* Report error. */
37564     }
37565     return(1);
37566 }
```

37567 The following demonstrates how the REG_NOTBOL flag could be used with *regex()* to find all
 37568 substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very
 37569 little error checking is done.)

```

37570     (void) regcomp (&re, pattern, 0);
37571     /* This call to regexec() finds the first match on the line. */
37572     error = regexec (&re, &buffer[0], 1, &pm, 0);
37573     while (error == 0) { /* While matches found. */
37574         /* Substring found between pm.rm_so and pm.rm_eo. */
37575         /* This call to regexec() finds the next match. */
37576         error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
37577     }

```

37578 APPLICATION USAGE

37579 An application could use:

```

37580     regerror(code, preg, (char *)NULL, (size_t)0)

```

37581 to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the
 37582 string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed,
 37583 static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger
 37584 buffer if it finds that this is too small.

37585 To match a pattern as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
 37586 2.13, Pattern Matching Notation, use the *fnmatch()* function.

37587 RATIONALE

37588 The *regexec()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are
 37589 supplied by the application, even if some elements of *pmatch* do not correspond to
 37590 subexpressions in *pattern*. The application writer should note that there is probably no reason
 37591 for using a value of *nmatch* that is larger than *preg->re_nsub*+1.

37592 The REG_NEWLINE flag supports a use of RE matching that is needed in some applications like
 37593 text editors. In such applications, the user supplies an RE asking the application to find a line
 37594 that matches the given expression. An anchor in such an RE anchors at the beginning or end of
 37595 any line. Such an application can pass a sequence of <newline>-separated lines to *regexec()* as a
 37596 single long string and specify REG_NEWLINE to *regcomp()* to get the desired behavior. The
 37597 application must ensure that there are no explicit <newline>s in *pattern* if it wants to ensure that
 37598 any match occurs entirely within a single line.

37599 The REG_NEWLINE flag affects the behavior of *regexec()*, but it is in the *cflags* parameter to
 37600 *regcomp()* to allow flexibility of implementation. Some implementations will want to generate
 37601 the same compiled RE in *regcomp()* regardless of the setting of REG_NEWLINE and have
 37602 *regexec()* handle anchors differently based on the setting of the flag. Other implementations will
 37603 generate different compiled REs based on the REG_NEWLINE.

37604 The REG_ICASE flag supports the operations taken by the *grep -i* option and the historical
 37605 implementations of *ex* and *vi*. Including this flag will make it easier for application code to be
 37606 written that does the same thing as these utilities.

37607 The substrings reported in *pmatch[]* are defined using offsets from the start of the string rather
 37608 than pointers. Since this is a new interface, there should be no impact on historical
 37609 implementations or applications, and offsets should be just as easy to use as pointers. The
 37610 change to offsets was made to facilitate future extensions in which the string to be searched is
 37611 presented to *regexec()* in blocks, allowing a string to be searched that is not all in memory at
 37612 once.

37613 The type **regoff_t** is used for the elements of *pmatch[]* to ensure that the application can
 37614 represent either the largest possible array in memory (important for an application conforming
 37615 to the Shell and Utilities volume of IEEE Std 1003.1-2001) or the largest possible file (important
 37616 for an application using the extension where a file is searched in chunks).

37617 The standard developers rejected the inclusion of a *regsub()* function that would be used to do
37618 substitutions for a matched RE. While such a routine would be useful to some applications, its
37619 utility would be much more limited than the matching function described here. Both RE parsing
37620 and substitution are possible to implement without support other than that required by the
37621 ISO C standard, but matching is much more complex than substituting. The only difficult part of
37622 substitution, given the information supplied by *regexexec()*, is finding the next character in a string
37623 when there can be multi-byte characters. That is a much larger issue, and one that needs a more
37624 general solution.

37625 The *errno* variable has not been used for error returns to avoid filling the *errno* name space for
37626 this feature.

37627 The interface is defined so that the matched substrings *rm_sp* and *rm_ep* are in a separate
37628 **regmatch_t** structure instead of in **regex_t**. This allows a single compiled RE to be used
37629 simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple
37630 threads of lightweight processes. (The *preg* argument to *regexexec()* is declared with type **const**, so
37631 the implementation is not permitted to use the structure to store intermediate results.) It also
37632 allows an application to request an arbitrary number of substrings from an RE. The number of
37633 subexpressions in the RE is reported in *re_nsub* in *preg*. With this change to *regexexec()*,
37634 consideration was given to dropping the REG_NOSUB flag since the user can now specify this
37635 with a zero *nmatch* argument to *regexexec()*. However, keeping REG_NOSUB allows an
37636 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()*
37637 that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if
37638 *nmatch* is not zero and if REG_NOSUB is not specified. Note that the **size_t** type, as defined in
37639 the ISO C standard, is unsigned, so the description of *regexexec()* does not need to address
37640 negative values of *nmatch*.

37641 REG_NOTBOL was added to allow an application to do repeated searches for the same pattern
37642 in a line. If the pattern contains a circumflex character that should match the beginning of a line,
37643 then the pattern should only match when matched against the beginning of the line. Without
37644 the REG_NOTBOL flag, the application could rewrite the expression for subsequent matches,
37645 but in the general case this would require parsing the expression. The need for REG_NOTEOL is
37646 not as clear; it was added for symmetry.

37647 The addition of the *regerror()* function addresses the historical need for conforming application
37648 programs to have access to error information more than “Function failed to compile/match your
37649 RE for unknown reasons”.

37650 This interface provides for two different methods of dealing with error conditions. The specific
37651 error codes (REG_EBRACE, for example), defined in **<regex.h>**, allow an application to recover
37652 from an error if it is so able. Many applications, especially those that use patterns supplied by a
37653 user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-
37654 readable error message to present to the user.

37655 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating
37656 memory to hold the generated string. The scheme used by *strerror()* in the ISO C standard was
37657 considered unacceptable since it creates difficulties for multi-threaded applications.

37658 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more
37659 descriptive message than would be possible with *errcode* alone. An implementation might, for
37660 example, save the character offset of the offending character of the pattern in a field of *preg*, and
37661 then include that in the generated message string. The implementation may also ignore *preg*.

37662 A REG_FILENAME flag was considered, but omitted. This flag caused *regexexec()* to match
37663 patterns as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13,
37664 Pattern Matching Notation instead of REs. This service is now provided by the *fnmatch()*

37665 function.

37666 Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and
37667 IEEE Std 1003.1-2001 in how to handle a “bad” regular expression. The ISO POSIX-2:1993
37668 standard says that many bad constructs “produce undefined results”, or that “the interpretation
37669 is undefined”. IEEE Std 1003.1-2001, however, says that the interpretation of such REs is
37670 unspecified. The term “undefined” means that the action by the application is an error, of
37671 similar severity to passing a bad pointer to a function.

37672 The *regcomp()* and *regexexec()* functions are required to accept any null-terminated string as the
37673 *pattern* argument. If the meaning of the string is “undefined”, the behavior of the function is
37674 “unspecified”. IEEE Std 1003.1-2001 does not specify how the functions will interpret the
37675 pattern; they might return error codes, or they might do pattern matching in some completely
37676 unexpected way, but they should not do something like abort the process.

37677 FUTURE DIRECTIONS

37678 None.

37679 SEE ALSO

37680 *fnmatch()*, *glob()*, Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13, Pattern
37681 Matching Notation, Base Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular
37682 Expressions, **<regex.h>**, **<sys/types.h>**

37683 CHANGE HISTORY

37684 First released in Issue 4. Derived from the ISO POSIX-2 standard.

37685 Issue 5

37686 Moved from POSIX2 C-language Binding to BASE.

37687 Issue 6

37688 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

37689 The following new requirements on POSIX implementations derive from alignment with the
37690 Single UNIX Specification:

- 37691 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
37692 required for conforming implementations of previous POSIX specifications, it was not
37693 required for UNIX applications.

37694 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

37695 The REG_ENOSYS constant is removed.

37696 The **restrict** keyword is added to the *regcomp()*, *regerror()*, and *regexexec()* prototypes for
37697 alignment with the ISO / IEC 9899:1999 standard.

37698 **NAME**

37699 remainder, remainderf, remainderl — remainder function

37700 **SYNOPSIS**

37701 #include <math.h>

37702 double remainder(double x, double y);

37703 float remainderf(float x, float y);

37704 long double remainderl(long double x, long double y);

37705 **DESCRIPTION**

37706 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37707 conflict between the requirements described here and the ISO C standard is unintentional. This
 37708 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37709 These functions shall return the floating-point remainder $r=x-ny$ when y is non-zero. The value
 37710 n is the integral value nearest the exact value x/y . When $|n-x/y| = 1/2$, the value n is chosen to
 37711 be even.

37712 The behavior of *remainder()* shall be independent of the rounding mode.37713 **RETURN VALUE**

37714 Upon successful completion, these functions shall return the floating-point remainder $r=x-ny$
 37715 when y is non-zero.

37716 MX If x or y is NaN, a NaN shall be returned.

37717 If x is infinite or y is 0 and the other is non-NaN, a domain error shall occur, and either a NaN (if
 37718 supported), or an implementation-defined value shall be returned.

37719 **ERRORS**

37720 These functions shall fail if:

37721 MX	Domain Error	The x argument is $\pm\text{Inf}$, or the y argument is ± 0 and the other argument is non-NaN.

37723		If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
37724		then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling
37725		& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
37726		shall be raised.

37727 **EXAMPLES**

37728 None.

37729 **APPLICATION USAGE**

37730 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 37731 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

37732 **RATIONALE**

37733 None.

37734 **FUTURE DIRECTIONS**

37735 None.

37736 **SEE ALSO**

37737 *abs()*, *div()*, *feclearexcept()*, *fetestexcept()*, *ldiv()*, the Base Definitions volume of
 37738 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
 37739 <math.h>

37740 **CHANGE HISTORY**

37741 First released in Issue 4, Version 2.

37742 **Issue 5**

37743 Moved from X/OPEN UNIX extension to BASE.

37744 **Issue 6**

37745 The *remainder()* function is no longer marked as an extension.

37746 The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999
37747 standard.

37748 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
37749 revised to align with the ISO/IEC 9899:1999 standard.

37750 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
37751 marked.

37752 **NAME**

37753 remove — remove a file

37754 **SYNOPSIS**

37755 #include <stdio.h>

37756 int remove(const char *path);

37757 **DESCRIPTION**

37758 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37759 conflict between the requirements described here and the ISO C standard is unintentional. This
 37760 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37761 The *remove()* function shall cause the file named by the pathname pointed to by *path* to be no
 37762 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,
 37763 unless it is created anew.

37764 CX If *path* does not name a directory, *remove(path)* shall be equivalent to *unlink(path)*.

37765 If *path* names a directory, *remove(path)* shall be equivalent to *rmdir(path)*.

37766 **RETURN VALUE**37767 CX Refer to *rmdir()* or *unlink()*.37768 **ERRORS**37769 CX Refer to *rmdir()* or *unlink()*.37770 **EXAMPLES**37771 **Removing Access to a File**37772 The following example shows how to remove access to a file named */home/cnd/old_mods*.

37773 #include <stdio.h>

37774 int status;

37775 ...

37776 status = remove("/home/cnd/old_mods");

37777 **APPLICATION USAGE**

37778 None.

37779 **RATIONALE**

37780 None.

37781 **FUTURE DIRECTIONS**

37782 None.

37783 **SEE ALSO**37784 *rmdir()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>37785 **CHANGE HISTORY**

37786 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C
 37787 standard.

37788 **Issue 6**

37789 Extensions beyond the ISO C standard are marked.

37790 The following new requirements on POSIX implementations derive from alignment with the
 37791 Single UNIX Specification:

37792
37793
37794

- The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to *rmdir()*.

37795 **NAME**

37796 remque — remove an element from a queue

37797 **SYNOPSIS**

37798 xSI #include <search.h>

37799 void remque(void *element);

37800

37801 **DESCRIPTION**37802 Refer to *insque()*.

37803 **NAME**

37804 remquo, remquof, remquol — remainder functions

37805 **SYNOPSIS**

37806 #include <math.h>

37807 double remquo(double x, double y, int *quo);

37808 float remquof(float x, float y, int *quo);

37809 long double remquol(long double x, long double y, int *quo);

37810 **DESCRIPTION**

37811 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37812 conflict between the requirements described here and the ISO C standard is unintentional. This
 37813 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37814 The *remquo()*, *remquof()*, and *remquol()* functions shall compute the same remainder as the
 37815 *remainder()*, *remainderf()*, and *remainderl()* functions, respectively. In the object pointed to by
 37816 *quo*, they store a value whose sign is the sign of x/y and whose magnitude is congruent modulo
 37817 2^n to the magnitude of the integral quotient of x/y , where n is an implementation-defined
 37818 integer greater than or equal to 3.

37819 An application wishing to check for error situations should set *errno* to zero and call
 37820 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 37821 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 37822 zero, an error has occurred.

37823 **RETURN VALUE**37824 These functions shall return $x \text{ REM } y$.37825 MX If x or y is NaN, a NaN shall be returned.

37826 If x is $\pm\text{Inf}$ or y is zero and the other argument is non-NaN, a domain error shall occur, and either
 37827 a NaN (if supported), or an implementation-defined value shall be returned.

37828 **ERRORS**

37829 These functions shall fail if:

37830 MX	Domain Error	The x argument is $\pm\text{Inf}$, or the y argument is ± 0 and the other argument is non-NaN.

37832	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.
37833	
37834	
37835	

37836 **EXAMPLES**

37837 None.

37838 **APPLICATION USAGE**

37839 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 37840 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

37841 **RATIONALE**

37842 These functions are intended for implementing argument reductions which can exploit a few
 37843 low-order bits of the quotient. Note that x may be so large in magnitude relative to y that an
 37844 exact representation of the quotient is not practical.

37845 **FUTURE DIRECTIONS**

37846 None.

37847 **SEE ALSO**

37848 *feclearexcept()*, *fetetestexcept()*, *remainder()*, the Base Definitions volume of IEEE Std 1003.1-2001,
37849 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

37850 **CHANGE HISTORY**

37851 First released in Issue 6. Derived from the ISO / IEC 9899:1999 standard.

37852 **NAME**

37853 rename — rename a file

37854 **SYNOPSIS**

37855 #include <stdio.h>

37856 int rename(const char *old, const char *new);

37857 **DESCRIPTION**

37858 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37859 conflict between the requirements described here and the ISO C standard is unintentional. This
 37860 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37861 The *rename()* function shall change the name of a file. The *old* argument points to the pathname
 37862 of the file to be renamed. The *new* argument points to the new pathname of the file.

37863 CX If either the *old* or *new* argument names a symbolic link, *rename()* shall operate on the symbolic
 37864 link itself, and shall not resolve the last component of the argument. If the *old* argument and the
 37865 *new* argument resolve to the same existing file, *rename()* shall return successfully and perform no
 37866 other action.

37867 If the *old* argument points to the pathname of a file that is not a directory, the *new* argument shall
 37868 not point to the pathname of a directory. If the link named by the *new* argument exists, it shall be
 37869 removed and *old* renamed to *new*. In this case, a link named *new* shall remain visible to other
 37870 processes throughout the renaming operation and refer either to the file referred to by *new* or *old*
 37871 before the operation began. Write access permission is required for both the directory containing
 37872 *old* and the directory containing *new*.

37873 If the *old* argument points to the pathname of a directory, the *new* argument shall not point to the
 37874 pathname of a file that is not a directory. If the directory named by the *new* argument exists, it
 37875 shall be removed and *old* renamed to *new*. In this case, a link named *new* shall exist throughout
 37876 the renaming operation and shall refer either to the directory referred to by *new* or *old* before the
 37877 operation began. If *new* names an existing directory, it shall be required to be an empty directory.

37878 If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed.
 37879 If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.

37880 The *new* pathname shall not contain a path prefix that names *old*. Write access permission is
 37881 required for the directory containing *old* and the directory containing *new*. If the *old* argument
 37882 points to the pathname of a directory, write access permission may be required for the directory
 37883 named by *old*, and, if it exists, the directory named by *new*.

37884 If the link named by the *new* argument exists and the file's link count becomes 0 when it is
 37885 removed and no process has the file open, the space occupied by the file shall be freed and the
 37886 file shall no longer be accessible. If one or more processes have the file open when the last link is
 37887 removed, the link shall be removed before *rename()* returns, but the removal of the file contents
 37888 shall be postponed until all references to the file are closed.

37889 Upon successful completion, *rename()* shall mark for update the *st_ctime* and *st_mtime* fields of
 37890 the parent directory of each file.

37891 If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be
 37892 unaffected.

37893 **RETURN VALUE**

37894 CX Upon successful completion, *rename()* shall return 0; otherwise, -1 shall be returned, *errno* shall
 37895 be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be
 37896 changed or created.

37897 **ERRORS**

37898 The *rename()* function shall fail if:

37899 CX	[EACCES]	A component of either path prefix denies search permission; or one of the directories containing <i>old</i> or <i>new</i> denies write permissions; or, write permission is required and is denied for a directory pointed to by the <i>old</i> or <i>new</i> arguments.
37900		
37901		
37902		
37903 CX	[EBUSY]	The directory named by <i>old</i> or <i>new</i> is currently in use by the system or another process, and the implementation considers this an error.
37904		
37905 CX	[EEXIST] or [ENOTEMPTY]	
37906		The link named by <i>new</i> is a directory that is not an empty directory.
37907 CX	[EINVAL]	The <i>new</i> directory pathname contains a path prefix that names the <i>old</i> directory.
37908		
37909 CX	[EIO]	A physical I/O error has occurred.
37910 CX	[EISDIR]	The <i>new</i> argument points to a directory and the <i>old</i> argument points to a file that is not a directory.
37911		
37912 CX	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
37913		
37914 CX	[EMLINK]	The file named by <i>old</i> is a directory, and the link count of the parent directory of <i>new</i> would exceed {LINK_MAX}.
37915		
37916 CX	[ENAMETOOLONG]	
37917		The length of the <i>old</i> or <i>new</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
37918		
37919 CX	[ENOENT]	The link named by <i>old</i> does not name an existing file, or either <i>old</i> or <i>new</i> points to an empty string.
37920		
37921 CX	[ENOSPC]	The directory that would contain <i>new</i> cannot be extended.
37922 CX	[ENOTDIR]	A component of either path prefix is not a directory; or the <i>old</i> argument names a directory and <i>new</i> argument names a non-directory file.
37923		
37924 XSI	[EPERM] or [EACCES]	
37925		The S_ISVTX flag is set on the directory containing the file referred to by <i>old</i> and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges; or <i>new</i> refers to an existing file, the S_ISVTX flag is set on the directory containing this file, and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges.
37926		
37927		
37928		
37929		
37930		
37931 CX	[EROFS]	The requested operation requires writing in a directory on a read-only file system.
37932		
37933 CX	[EXDEV]	The links named by <i>new</i> and <i>old</i> are on different file systems and the implementation does not support links between file systems.
37934		
37935		The <i>rename()</i> function may fail if:
37936 XSI	[EBUSY]	The file named by the <i>old</i> or <i>new</i> arguments is a named STREAM.
37937 CX	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
37938		

37939 CX [ENAMETOOLONG]
 37940 As a result of encountering a symbolic link in resolution of the *path* argument,
 37941 the length of the substituted pathname string exceeded {PATH_MAX}.

37942 CX [ETXTBSY] The file to be renamed is a pure procedure (shared text) file that is being
 37943 executed.

37944 EXAMPLES

37945 Renaming a File

37946 The following example shows how to rename a file named **/home/cnd/mod1** to
 37947 **/home/cnd/mod2**.

```
37948 #include <stdio.h>
37949 int status;
37950 ...
37951 status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

37952 APPLICATION USAGE

37953 Some implementations mark for update the *st_ctime* field of renamed files and some do not.
 37954 Applications which make use of the *st_ctime* field may behave differently with respect to
 37955 renamed files unless they are designed to allow for either behavior.

37956 RATIONALE

37957 This *rename()* function is equivalent for regular files to that defined by the ISO C standard. Its
 37958 inclusion here expands that definition to include actions on directories and specifies behavior
 37959 when the *new* parameter names a file that already exists. That specification requires that the
 37960 action of the function be atomic.

37961 One of the reasons for introducing this function was to have a means of renaming directories
 37962 while permitting implementations to prohibit the use of *link()* and *unlink()* with directories,
 37963 thus constraining links to directories to those made by *mkdir()*.

37964 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
37965 rename("x", "x");
37966 does not remove the file.
```

37967 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

37968 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir()* and [EBUSY] in
 37969 *unlink()*. For a discussion of [EXDEV], see *link()*.

37970 FUTURE DIRECTIONS

37971 None.

37972 SEE ALSO

37973 *link()*, *rmdir()*, *symlink()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 37974 **<stdio.h>**

37975 CHANGE HISTORY

37976 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

37977 Issue 5

37978 The [EBUSY] error is added to the “may fail” part of the ERRORS section.

37979 **Issue 6**

37980 Extensions beyond the ISO C standard are marked.

37981 The following new requirements on POSIX implementations derive from alignment with the
37982 Single UNIX Specification:

- 37983 • The [EIO] mandatory error condition is added.
- 37984 • The [ELOOP] mandatory error condition is added.
- 37985 • A second [ENAMETOOLONG] is added as an optional error condition.
- 37986 • The [ETXTBSY] optional error condition is added.

37987 The following changes were made to align with the IEEE P1003.1a draft standard:

- 37988 • Details are added regarding the treatment of symbolic links.
- 37989 • The [ELOOP] optional error condition is added.

37990 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

37991 **NAME**

37992 rewind — reset the file position indicator in a stream

37993 **SYNOPSIS**

37994 #include <stdio.h>

37995 void rewind(FILE *stream);

37996 **DESCRIPTION**

37997 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37998 conflict between the requirements described here and the ISO C standard is unintentional. This
37999 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38000 The call:

38001 rewind(stream)

38002 shall be equivalent to:

38003 (void) fseek(stream, 0L, SEEK_SET)

38004 except that *rewind()* shall also clear the error indicator.

38005 CX Since *rewind()* does not return a value, an application wishing to detect errors should clear *errno*,
38006 then call *rewind()*, and if *errno* is non-zero, assume an error has occurred.

38007 **RETURN VALUE**

38008 The *rewind()* function shall not return a value.

38009 **ERRORS**

38010 CX Refer to *fseek()* with the exception of [EINVAL] which does not apply.

38011 **EXAMPLES**

38012 None.

38013 **APPLICATION USAGE**

38014 None.

38015 **RATIONALE**

38016 None.

38017 **FUTURE DIRECTIONS**

38018 None.

38019 **SEE ALSO**

38020 *fseek()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

38021 **CHANGE HISTORY**

38022 First released in Issue 1. Derived from Issue 1 of the SVID.

38023 **Issue 6**

38024 Extensions beyond the ISO C standard are marked.

38025 **NAME**

38026 `rewinddir` — reset the position of a directory stream to the beginning of a directory

38027 **SYNOPSIS**

38028 `#include <dirent.h>`

38029 `void rewinddir(DIR *dirp);`

38030 **DESCRIPTION**

38031 The `rewinddir()` function shall reset the position of the directory stream to which *dirp* refers to
38032 the beginning of the directory. It shall also cause the directory stream to refer to the current state
38033 of the corresponding directory, as a call to `opendir()` would have done. If *dirp* does not refer to a
38034 directory stream, the effect is undefined.

38035 After a call to the `fork()` function, either the parent or child (but not both) may continue
38036 xsi processing the directory stream using `readdir()`, `rewinddir()`, or `seekdir()`. If both the parent and
38037 child processes use these functions, the result is undefined.

38038 **RETURN VALUE**

38039 The `rewinddir()` function shall not return a value.

38040 **ERRORS**

38041 No errors are defined.

38042 **EXAMPLES**

38043 None.

38044 **APPLICATION USAGE**

38045 The `rewinddir()` function should be used in conjunction with `opendir()`, `readdir()`, and `closedir()` to
38046 examine the contents of the directory. This method is recommended for portability.

38047 **RATIONALE**

38048 None.

38049 **FUTURE DIRECTIONS**

38050 None.

38051 **SEE ALSO**

38052 `closedir()`, `opendir()`, `readdir()`, the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**
38053 **<sys/types.h>**

38054 **CHANGE HISTORY**

38055 First released in Issue 2.

38056 **Issue 6**

38057 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

38058 The following new requirements on POSIX implementations derive from alignment with the
38059 Single UNIX Specification:

- 38060 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
38061 required for conforming implementations of previous POSIX specifications, it was not
38062 required for UNIX applications.

38063 **NAME**38064 rindex — character string operations (**LEGACY**)38065 **SYNOPSIS**

38066 xSI #include <strings.h>

38067 char *rindex(const char *s, int c);

38068

38069 **DESCRIPTION**38070 The *rindex()* function shall be equivalent to *strrchr()*.38071 **RETURN VALUE**38072 Refer to *strrchr()*.38073 **ERRORS**38074 Refer to *strrchr()*.38075 **EXAMPLES**

38076 None.

38077 **APPLICATION USAGE**38078 The *strrchr()* function is preferred over this function.38079 For maximum portability, it is recommended to replace the function call to *rindex()* as follows:

38080 #define rindex(a,b) strrchr((a),(b))

38081 **RATIONALE**

38082 None.

38083 **FUTURE DIRECTIONS**

38084 This function may be withdrawn in a future version.

38085 **SEE ALSO**38086 *strrchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <strings.h>38087 **CHANGE HISTORY**

38088 First released in Issue 4, Version 2.

38089 **Issue 5**

38090 Moved from X/OPEN UNIX extension to BASE.

38091 **Issue 6**

38092 This function is marked LEGACY.

38093 **NAME**

38094 rint, rintf, rintl — round-to-nearest integral value

38095 **SYNOPSIS**

38096 #include <math.h>

38097 double rint(double x);

38098 float rintf(float x);

38099 long double rintl(long double x);

38100 **DESCRIPTION**

38101 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38102 conflict between the requirements described here and the ISO C standard is unintentional. This
 38103 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38104 These functions shall return the integral value (represented as a **double**) nearest x in the
 38105 direction of the current rounding mode. The current rounding mode is implementation-defined.

38106 If the current rounding mode rounds toward negative infinity, then *rint()* shall be equivalent to
 38107 *floor()*. If the current rounding mode rounds toward positive infinity, then *rint()* shall be
 38108 equivalent to *ceil()*.

38109 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that
 38110 they may raise the inexact floating-point exception if the result differs in value from the
 38111 argument.

38112 An application wishing to check for error situations should set *errno* to zero and call
 38113 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 38114 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 38115 zero, an error has occurred.

38116 **RETURN VALUE**

38117 Upon successful completion, these functions shall return the integer (represented as a double
 38118 precision number) nearest x in the direction of the current rounding mode.

38119 MX If x is NaN, a NaN shall be returned.

38120 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

38121 XSI If the correct value would cause overflow, a range error shall occur and *rint()*, *rintf()*, and *rintl()*
 38122 shall return the value of the macro $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and $\pm \text{HUGE\_VALL}$ (with the
 38123 same sign as x), respectively.

38124 **ERRORS**

38125 These functions shall fail if:

38126 XSI **Range Error** The result would cause an overflow.

38127 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 38128 then *errno* shall be set to [ERANGE]. If the integer expression
 38129 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 38130 floating-point exception shall be raised.

38131 **EXAMPLES**

38132 None.

38133 **APPLICATION USAGE**

38134 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
38135 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38136 **RATIONALE**

38137 None.

38138 **FUTURE DIRECTIONS**

38139 None.

38140 **SEE ALSO**

38141 *abs()*, *ceil()*, *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, *nearbyint()*, the Base Definitions volume
38142 of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
38143 <math.h>

38144 **CHANGE HISTORY**

38145 First released in Issue 4, Version 2.

38146 **Issue 5**

38147 Moved from X/OPEN UNIX extension to BASE.

38148 **Issue 6**

38149 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 38150 • The *rintf()* and *rintl()* functions are added.
- 38151 • The *rint()* function is no longer marked as an extension.
- 38152 • The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
38153 revised to align with the ISO/IEC 9899:1999 standard.
- 38154 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
38155 marked.

38156 NAME

38157 rmdir — remove a directory

38158 SYNOPSIS

38159 #include <unistd.h>

38160 int rmdir(const char *path);

38161 DESCRIPTION

38162 The *rmdir()* function shall remove a directory whose name is given by *path*. The directory shall
38163 be removed only if it is an empty directory.

38164 If the directory is the root directory or the current working directory of any process, it is
38165 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].

38166 If *path* names a symbolic link, then *rmdir()* shall fail and set *errno* to [ENOTDIR].

38167 If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir()* shall
38168 fail.

38169 If the directory's link count becomes 0 and no process has the directory open, the space occupied
38170 by the directory shall be freed and the directory shall no longer be accessible. If one or more
38171 processes have the directory open when the last link is removed, the dot and dot-dot entries, if
38172 present, shall be removed before *rmdir()* returns and no new entries may be created in the
38173 directory, but the directory shall not be removed until all references to the directory are closed.

38174 If the directory is not an empty directory, *rmdir()* shall fail and set *errno* to [EEXIST] or
38175 [ENOTEMPTY].

38176 Upon successful completion, the *rmdir()* function shall mark for update the *st_ctime* and
38177 *st_mtime* fields of the parent directory.

38178 RETURN VALUE

38179 Upon successful completion, the function *rmdir()* shall return 0. Otherwise, -1 shall be returned,
38180 and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.

38181 ERRORS

38182 The *rmdir()* function shall fail if:

38183 [EACCES] Search permission is denied on a component of the path prefix, or write
38184 permission is denied on the parent directory of the directory to be removed.

38185 [EBUSY] The directory to be removed is currently in use by the system or some process
38186 and the implementation considers this to be an error.

38187 [EEXIST] or [ENOTEMPTY]

38188 The *path* argument names a directory that is not an empty directory, or there
38189 are hard links to the directory other than dot or a single entry in dot-dot.

38190 [EINVAL] The *path* argument contains a last component that is dot.

38191 [EIO] A physical I/O error has occurred.

38192 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
38193 argument.

38194 [ENAMETOOLONG]

38195 The length of the *path* argument exceeds {PATH_MAX} or a pathname
38196 component is longer than {NAME_MAX}.

38197 [ENOENT] A component of *path* does not name an existing file, or the *path* argument
38198 names a nonexistent directory or points to an empty string.

38199 [ENOTDIR] A component of *path* is not a directory.

38200 xsi [EPERM] or [EACCES]
 38201 The S_ISVTX flag is set on the parent directory of the directory to be removed
 38202 and the caller is not the owner of the directory to be removed, nor is the caller
 38203 the owner of the parent directory, nor does the caller have the appropriate
 38204 privileges.

38205 [EROFS] The directory entry to be removed resides on a read-only file system.

38206 The *rmdir()* function may fail if:

38207 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 38208 resolution of the *path* argument.

38209 [ENAMETOOLONG]
 38210 As a result of encountering a symbolic link in resolution of the *path* argument,
 38211 the length of the substituted pathname string exceeded {PATH_MAX}.

38212 EXAMPLES

38213 Removing a Directory

38214 The following example shows how to remove a directory named **/home/cnd/mod1**.

```
38215 #include <unistd.h>
38216 int status;
38217 ...
38218 status = rmdir("/home/cnd/mod1");
```

38219 APPLICATION USAGE

38220 None.

38221 RATIONALE

38222 The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the
 38223 condition when the directory to be removed does not exist or *new* already exists. When the 1984
 38224 /usr/group standard was published, it contained [EEXIST] instead. When these functions were
 38225 adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several
 38226 existing applications and implementations support/use both forms, and no agreement could be
 38227 reached on either value. All implementations are required to supply both [EEXIST] and
 38228 [ENOTEMPTY] in **<errno.h>** with distinct values, so that applications can use both values in C-
 38229 language **case** statements.

38230 The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the
 38231 parent directory to be removed is not clear, particularly in the presence of multiple links to a
 38232 directory.

38233 The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir()* when there are
 38234 multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or
 38235 [ENOTEMPTY] clarifies the behavior in this case.

38236 If the process' current working directory is being removed, that should be an allowed error.

38237 Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in
 38238 Section 2.3 (on page 21) about returning any one of the possible errors permits that behavior to
 38239 continue. The [ELOOP] error may be returned if more than {SYMLOOP_MAX} symbolic links
 38240 are encountered during resolution of the *path* argument.

38241 **FUTURE DIRECTIONS**

38242 None.

38243 **SEE ALSO**

38244 Section 2.3 (on page 21), *mkdir()*, *remove()*, *unlink()*, the Base Definitions volume of
38245 IEEE Std 1003.1-2001, <**unistd.h**>

38246 **CHANGE HISTORY**

38247 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

38248 **Issue 6**

38249 The following new requirements on POSIX implementations derive from alignment with the
38250 Single UNIX Specification:

- 38251 • The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- 38252 • The [EIO] mandatory error condition is added.
- 38253 • The [ELOOP] mandatory error condition is added.
- 38254 • A second [ENAMETOOLONG] is added as an optional error condition.

38255 The following changes were made to align with the IEEE P1003.1a draft standard:

- 38256 • The [ELOOP] optional error condition is added.

38257 **NAME**

38258 round, roundf, roundl — round to the nearest integer value in a floating-point format

38259 **SYNOPSIS**

38260 #include <math.h>

38261 double round(double x);

38262 float roundf(float x);

38263 long double roundl(long double x);

38264 **DESCRIPTION**

38265 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38266 conflict between the requirements described here and the ISO C standard is unintentional. This
 38267 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38268 These functions shall round their argument to the nearest integer value in floating-point format,
 38269 rounding halfway cases away from zero, regardless of the current rounding direction.

38270 An application wishing to check for error situations should set *errno* to zero and call
 38271 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 38272 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 38273 zero, an error has occurred.

38274 **RETURN VALUE**

38275 Upon successful completion, these functions shall return the rounded integer value.

38276 MX If *x* is NaN, a NaN shall be returned.38277 If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.

38278 XSI If the correct value would cause overflow, a range error shall occur and *round()*, *roundf()*, and
 38279 *roundl()* shall return the value of the macro $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and $\pm \text{HUGE\_VALL}$
 38280 (with the same sign as *x*), respectively.

38281 **ERRORS**

38282 These functions may fail if:

38283 XSI **Range Error** The result overflows.

38284 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 38285 then *errno* shall be set to [ERANGE]. If the integer expression
 38286 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 38287 floating-point exception shall be raised.

38288 **EXAMPLES**

38289 None.

38290 **APPLICATION USAGE**

38291 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 38292 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38293 **RATIONALE**

38294 None.

38295 **FUTURE DIRECTIONS**

38296 None.

38297 **SEE ALSO**

38298 *feclearexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,
38299 Treatment of Error Conditions for Mathematical Functions, **<math.h>**

38300 **CHANGE HISTORY**

38301 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38302 **NAME**

38303 scalb — load exponent of a radix-independent floating-point number

38304 **SYNOPSIS**

38305 OB XSI #include <math.h>

38306 double scalb(double x, double n);

38307

38308 **DESCRIPTION**

38309 The *scalb()* function shall compute $x \cdot r^n$, where r is the radix of the machine's floating-point arithmetic. When r is 2, *scalb()* shall be equivalent to *ldexp()*. The value of r is FLT_RADIX which is defined in <float.h>.

38312 An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

38316 **RETURN VALUE**38317 Upon successful completion, the *scalb()* function shall return $x \cdot r^n$.38318 If x or n is NaN, a NaN shall be returned.38319 If n is zero, x shall be returned.38320 If x is $\pm\text{Inf}$ and n is not $-\text{Inf}$, x shall be returned.38321 If x is ± 0 and n is not $+\text{Inf}$, x shall be returned.

38322 If x is ± 0 and n is $+\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

38324 If x is $\pm\text{Inf}$ and n is $-\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

38326 If the result would cause an overflow, a range error shall occur and $\pm\text{HUGE\_VAL}$ (according to the sign of x) shall be returned.

38328 If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

38330 If the correct value would cause underflow, and is not representable, a range error may occur, and 0.0 shall be returned.

38332 **ERRORS**38333 The *scalb()* function shall fail if:38334 Domain Error If x is zero and n is $+\text{Inf}$, or x is Inf and n is $-\text{Inf}$.

38335 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

38339 Range Error The result would overflow.

38340 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

38344 The *scalb()* function may fail if:

38345 Range Error The result underflows.

38346 If the integer expression (*math_errhandling* & *MATH_ERRNO*) is non-zero,
 38347 then *errno* shall be set to [ERANGE]. If the integer expression
 38348 (*math_errhandling* & *MATH_ERREXCEPT*) is non-zero, then the underflow
 38349 floating-point exception shall be raised.

38350 EXAMPLES

38351 None.

38352 APPLICATION USAGE

38353 Applications should use either *scalbln()*, *scalblnf()*, or *scalblnl()* in preference to this function.

38354 IEEE Std 1003.1-2001 only defines the behavior for the *scalb()* function when the *n* argument is
 38355 an integer, a NaN, or Inf. The behavior of other values for the *n* argument is unspecified.

38356 On error, the expressions (*math_errhandling* & *MATH_ERRNO*) and (*math_errhandling* &
 38357 *MATH_ERREXCEPT*) are independent of each other, but at least one of them must be non-zero.

38358 RATIONALE

38359 None.

38360 FUTURE DIRECTIONS

38361 None.

38362 SEE ALSO

38363 *feclearexcept()*, *fetestexcept()*, *ilogb()*, *ldexp()*, *logb()*, *scalbln()*, the Base Definitions volume of
 38364 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
 38365 <float.h>, <math.h>

38366 CHANGE HISTORY

38367 First released in Issue 4, Version 2.

38368 Issue 5

38369 Moved from X/OPEN UNIX extension to BASE.

38370 The DESCRIPTION is updated to indicate how an application should check for an error. This
 38371 text was previously published in the APPLICATION USAGE section.

38372 Issue 6

38373 This function is marked obsolescent.

38374 Although this function is not part of the ISO/IEC 9899:1999 standard, the RETURN VALUE and
 38375 ERRORS sections are updated to align with the error handling in the ISO/IEC 9899:1999
 38376 standard.

38377 **NAME**

38378 scalbn, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT_RADIX |

38379 **SYNOPSIS**

38380 #include <math.h>

```

38381       double scalbn(double x, long n);
38382       float scalblnf(float x, long n);
38383       long double scalblnl(long double x, long n);
38384       double scalbn(double x, int n);
38385       float scalbnf(float x, int n);
38386       long double scalbnl(long double x, int n);

```

38387 **DESCRIPTION**

38388 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38389 conflict between the requirements described here and the ISO C standard is unintentional. This
 38390 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38391 These functions shall compute $x * \text{FLT\_RADIX}^n$ efficiently, not normally by computing
 38392 FLT_RADIX^n explicitly.

38393 An application wishing to check for error situations should set *errno* to zero and call
 38394 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 38395 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 38396 zero, an error has occurred.

38397 **RETURN VALUE**38398 Upon successful completion, these functions shall return $x * \text{FLT\_RADIX}^n$.

38399 If the result would cause overflow, a range error shall occur and these functions shall return
 38400 $\pm\text{HUGE\_VAL}$, $\pm\text{HUGE\_VALF}$, and $\pm\text{HUGE\_VALL}$ (according to the sign of *x*) as appropriate for
 38401 the return type of the function.

38402 If the correct value would cause underflow, and is not representable, a range error may occur,
 38403 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

38404 MX If *x* is NaN, a NaN shall be returned.38405 If *x* is ± 0 or $\pm\text{Inf}$, *x* shall be returned.38406 If *n* is 0, *x* shall be returned.

38407 If the correct value would cause underflow, and is representable, a range error may occur and
 38408 the correct value shall be returned.

38409 **ERRORS**

38410 These functions shall fail if:

38411 Range Error The result overflows.

38412 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 38413 then *errno* shall be set to [ERANGE]. If the integer expression
 38414 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 38415 floating-point exception shall be raised.

38416 These functions may fail if:

38417 Range Error The result underflows.

38418 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 38419 then *errno* shall be set to [ERANGE]. If the integer expression

38420 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
38421 floating-point exception shall be raised.

38422 **EXAMPLES**

38423 None.

38424 **APPLICATION USAGE**

38425 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
38426 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38427 **RATIONALE**

38428 These functions are named so as to avoid conflicting with the historical definition of the *scalb()*
38429 function from the Single UNIX Specification. The difference is that the *scalb()* function has a
38430 second argument of **double** instead of **int**. The *scalb()* function is not part of the ISO C standard.
38431 The three functions whose second type is **long** are provided because the factor required to scale
38432 from the smallest positive floating-point value to the largest finite one, on many
38433 implementations, is too large to represent in the minimum-width **int** format.

38434 **FUTURE DIRECTIONS**

38435 None.

38436 **SEE ALSO**

38437 *feclearexcept()*, *fetestexcept()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section
38438 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

38439 **CHANGE HISTORY**

38440 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38441 **NAME**

38442 scanf — convert formatted input

38443 **SYNOPSIS**

38444 #include <stdio.h>

38445 int scanf(const char *restrict *format*, ...);

38446 **DESCRIPTION**

38447 Refer to *fscanf()*.

38448 **NAME**38449 sched_get_priority_max, sched_get_priority_min — get priority limits (**REALTIME**)38450 **SYNOPSIS**

38451 PS|TPS #include <sched.h>

38452 int sched_get_priority_max(int *policy*);38453 int sched_get_priority_min(int *policy*);

38454

38455 **DESCRIPTION**38456 The *sched_get_priority_max()* and *sched_get_priority_min()* functions shall return the appropriate
38457 maximum or minimum, respectively, for the scheduling policy specified by *policy*.38458 The value of *policy* shall be one of the scheduling policy values defined in <**sched.h**>.38459 **RETURN VALUE**38460 If successful, the *sched_get_priority_max()* and *sched_get_priority_min()* functions shall return the
38461 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a
38462 value of -1 and set *errno* to indicate the error.38463 **ERRORS**38464 The *sched_get_priority_max()* and *sched_get_priority_min()* functions shall fail if:38465 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling
38466 policy.38467 **EXAMPLES**

38468 None.

38469 **APPLICATION USAGE**

38470 None.

38471 **RATIONALE**

38472 None.

38473 **FUTURE DIRECTIONS**

38474 None.

38475 **SEE ALSO**38476 *sched_getparam()*, *sched_setparam()*, *sched_getscheduler()*, *sched_rr_get_interval()*,
38477 *sched_setscheduler()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sched.h**>38478 **CHANGE HISTORY**

38479 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38480 **Issue 6**

38481 These functions are marked as part of the Process Scheduling option.

38482 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38483 implementation does not support the Process Scheduling option.38484 The [ESRCH] error condition has been removed since these functions do not take a *pid*
38485 argument.38486 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin
38487 code in the SYNOPSIS to PS|TPS.

38488 NAME

38489 sched_getparam — get scheduling parameters (**REALTIME**)

38490 SYNOPSIS

38491 PS `#include <sched.h>`

38492 `int sched_getparam(pid_t pid, struct sched_param *param);`

38493

38494 DESCRIPTION

38495 The *sched_getparam()* function shall return the scheduling parameters of a process specified by
38496 *pid* in the **sched_param** structure pointed to by *param*.

38497 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
38498 parameters for the process whose process ID is equal to *pid* shall be returned.

38499 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of
38500 the *sched_getparam()* function is unspecified if the value of *pid* is negative.

38501 RETURN VALUE

38502 Upon successful completion, the *sched_getparam()* function shall return zero. If the call to
38503 *sched_getparam()* is unsuccessful, the function shall return a value of -1 and set *errno* to indicate
38504 the error.

38505 ERRORS

38506 The *sched_getparam()* function shall fail if:

38507 [EPERM] The requesting process does not have permission to obtain the scheduling
38508 parameters of the specified process.

38509 [ESRCH] No process can be found corresponding to that specified by *pid*.

38510 EXAMPLES

38511 None.

38512 APPLICATION USAGE

38513 None.

38514 RATIONALE

38515 None.

38516 FUTURE DIRECTIONS

38517 None.

38518 SEE ALSO

38519 *sched_getscheduler()*, *sched_setparam()*, *sched_setscheduler()*, the Base Definitions volume of
38520 IEEE Std 1003.1-2001, **<sched.h>**

38521 CHANGE HISTORY

38522 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38523 Issue 6

38524 The *sched_getparam()* function is marked as part of the Process Scheduling option.

38525 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38526 implementation does not support the Process Scheduling option.

38527 **NAME**38528 sched_getscheduler — get scheduling policy (**REALTIME**)38529 **SYNOPSIS**

38530 PS #include <sched.h>

38531 int sched_getscheduler(pid_t pid);

38532

38533 **DESCRIPTION**

38534 The *sched_getscheduler()* function shall return the scheduling policy of the process specified by
 38535 *pid*. If the value of *pid* is negative, the behavior of the *sched_getscheduler()* function is
 38536 unspecified.

38537 The values that can be returned by *sched_getscheduler()* are defined in the <**sched.h**> header.

38538 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 38539 policy shall be returned for the process whose process ID is equal to *pid*.

38540 If *pid* is zero, the scheduling policy shall be returned for the calling process.

38541 **RETURN VALUE**

38542 Upon successful completion, the *sched_getscheduler()* function shall return the scheduling policy
 38543 of the specified process. If unsuccessful, the function shall return -1 and set *errno* to indicate the
 38544 error.

38545 **ERRORS**

38546 The *sched_getscheduler()* function shall fail if:

38547 [EPERM] The requesting process does not have permission to determine the scheduling
 38548 policy of the specified process.

38549 [ESRCH] No process can be found corresponding to that specified by *pid*.

38550 **EXAMPLES**

38551 None.

38552 **APPLICATION USAGE**

38553 None.

38554 **RATIONALE**

38555 None.

38556 **FUTURE DIRECTIONS**

38557 None.

38558 **SEE ALSO**

38559 *sched_getparam()*, *sched_setparam()*, *sched_setscheduler()*, the Base Definitions volume of
 38560 IEEE Std 1003.1-2001, <**sched.h**>

38561 **CHANGE HISTORY**

38562 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38563 **Issue 6**

38564 The *sched_getscheduler()* function is marked as part of the Process Scheduling option.

38565 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 38566 implementation does not support the Process Scheduling option.

38567 NAME

38568 sched_rr_get_interval — get execution time limits (**REALTIME**)

38569 SYNOPSIS

38570 PS|TPS #include <sched.h>

38571 int sched_rr_get_interval(pid_t pid, struct timespec *interval);

38572

38573 DESCRIPTION

38574 The *sched_rr_get_interval()* function shall update the **timespec** structure referenced by the
38575 *interval* argument to contain the current execution time limit (that is, time quantum) for the
38576 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process
38577 shall be returned.

38578 RETURN VALUE

38579 If successful, the *sched_rr_get_interval()* function shall return zero. Otherwise, it shall return a
38580 value of -1 and set *errno* to indicate the error.

38581 ERRORS

38582 The *sched_rr_get_interval()* function shall fail if:

38583 [ESRCH] No process can be found corresponding to that specified by *pid*.

38584 EXAMPLES

38585 None.

38586 APPLICATION USAGE

38587 None.

38588 RATIONALE

38589 None.

38590 FUTURE DIRECTIONS

38591 None.

38592 SEE ALSO

38593 *sched_getparam()*, *sched_get_priority_max()*, *sched_getscheduler()*, *sched_setparam()*,
38594 *sched_setscheduler()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sched.h**>

38595 CHANGE HISTORY

38596 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38597 Issue 6

38598 The *sched_rr_get_interval()* function is marked as part of the Process Scheduling option.

38599 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38600 implementation does not support the Process Scheduling option.

38601 IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in
38602 the SYNOPSIS to PS|TPS.

38603 **NAME**38604 sched_setparam — set scheduling parameters (**REALTIME**)38605 **SYNOPSIS**38606 PS `#include <sched.h>`38607 `int sched_setparam(pid_t pid, const struct sched_param *param);`

38608

38609 **DESCRIPTION**

38610 The *sched_setparam()* function shall set the scheduling parameters of the process specified by *pid*
 38611 to the values specified by the **sched_param** structure pointed to by *param*. The value of the
 38612 *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive
 38613 priority range for the current scheduling policy of the process specified by *pid*. Higher
 38614 numerical values for the priority represent higher priorities. If the value of *pid* is negative, the
 38615 behavior of the *sched_setparam()* function is unspecified.

38616 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 38617 parameters shall be set for the process whose process ID is equal to *pid*.

38618 If *pid* is zero, the scheduling parameters shall be set for the calling process.

38619 The conditions under which one process has permission to change the scheduling parameters of
 38620 another process are implementation-defined.

38621 Implementations may require the requesting process to have the appropriate privilege to set its
 38622 own scheduling parameters or those of another process.

38623 The target process, whether it is running or not running, shall be moved to the tail of the thread
 38624 list for its priority.

38625 If the priority of the process specified by the *pid* argument is set higher than that of the lowest
 38626 priority running process and if the specified process is ready to run, the process specified by the
 38627 *pid* argument shall preempt a lowest priority running process. Similarly, if the process calling
 38628 *sched_setparam()* sets its own priority lower than that of one or more other non-empty process
 38629 lists, then the process that is the head of the highest priority list shall also preempt the calling
 38630 process. Thus, in either case, the originating process might not receive notification of the
 38631 completion of the requested priority change until the higher priority process has executed.

38632 ss If the scheduling policy of the target process is SCHED_SPORADIC, the value specified by the
 38633 *sched_ss_low_priority* member of the *param* argument shall be any integer within the inclusive
 38634 priority range for the sporadic server policy. The *sched_ss_repl_period* and *sched_ss_init_budget*
 38635 members of the *param* argument shall represent the time parameters to be used by the sporadic
 38636 server scheduling policy for the target process. The *sched_ss_max_repl* member of the *param*
 38637 argument shall represent the maximum number of replenishments that are allowed to be
 38638 pending simultaneously for the process scheduled under this scheduling policy.

38639 The specified *sched_ss_repl_period* shall be greater than or equal to the specified
 38640 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

38641 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
 38642 function to succeed; if not, the function shall fail.

38643 If the scheduling policy of the target process is either SCHED_FIFO or SCHED_RR, the
 38644 *sched_ss_low_priority*, *sched_ss_repl_period*, and *sched_ss_init_budget* members of the *param*
 38645 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process
 38646 is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the effects of these members are
 38647 implementation-defined; this case includes the SCHED_OTHER policy.

38648 If the current scheduling policy for the process specified by *pid* is not SCHED_FIFO,
 38649 ss SCHED_RR, or SCHED_SPORADIC, the result is implementation-defined; this case includes the
 38650 SCHED_OTHER policy.

38651 The effect of this function on individual threads is dependent on the scheduling contention
 38652 scope of the threads:

- 38653 • For threads with system scheduling contention scope, these functions shall have no effect on
 38654 their scheduling.
- 38655 • For threads with process scheduling contention scope, the threads' scheduling parameters
 38656 shall not be affected. However, the scheduling of these threads with respect to threads in
 38657 other processes may be dependent on the scheduling parameters of their process, which are
 38658 governed using these functions.

38659 If an implementation supports a two-level scheduling model in which library threads are
 38660 multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled
 38661 entities for the system contention scope threads shall not be affected by these functions.

38662 The underlying kernel-scheduled entities for the process contention scope threads shall have
 38663 their scheduling parameters changed to the value specified in *param*. Kernel-scheduled entities
 38664 for use by process contention scope threads that are created after this call completes shall inherit
 38665 their scheduling policy and associated scheduling parameters from the process.

38666 This function is not atomic with respect to other threads in the process. Threads may continue to
 38667 execute while this function call is in the process of changing the scheduling policy for the
 38668 underlying kernel-scheduled entities used by the process contention scope threads.

38669 RETURN VALUE

38670 If successful, the *sched_setparam()* function shall return zero.

38671 If the call to *sched_setparam()* is unsuccessful, the priority shall remain unchanged, and the
 38672 function shall return a value of -1 and set *errno* to indicate the error.

38673 ERRORS

38674 The *sched_setparam()* function shall fail if:

- 38675 [EINVAL] One or more of the requested scheduling parameters is outside the range
 38676 defined for the scheduling policy of the specified *pid*.
- 38677 [EPERM] The requesting process does not have permission to set the scheduling
 38678 parameters for the specified process, or does not have the appropriate
 38679 privilege to invoke *sched_setparam()*.
- 38680 [ESRCH] No process can be found corresponding to that specified by *pid*.

38681 EXAMPLES

38682 None.

38683 APPLICATION USAGE

38684 None.

38685 RATIONALE

38686 None.

38687 FUTURE DIRECTIONS

38688 None.

38689 **SEE ALSO**

38690 *sched_getparam()*, *sched_getscheduler()*, *sched_setscheduler()*, the Base Definitions volume of
38691 IEEE Std 1003.1-2001, <**sched.h**>

38692 **CHANGE HISTORY**

38693 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38694 **Issue 6**

38695 The *sched_setparam()* function is marked as part of the Process Scheduling option.

38696 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38697 implementation does not support the Process Scheduling option.

38698 The following new requirements on POSIX implementations derive from alignment with the
38699 Single UNIX Specification:

38700 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
38701 added.

38702 • Sections describing two-level scheduling and atomicity of the function are added.

38703 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

38704 IEEE PASC Interpretation 1003.1 #100 is applied.

38705 NAME

38706 sched_setscheduler — set scheduling policy and parameters (**REALTIME**)

38707 SYNOPSIS

38708 PS `#include <sched.h>`

38709 `int sched_setscheduler(pid_t pid, int policy,`
38710 `const struct sched_param *param);`

38711

38712 DESCRIPTION

38713 The *sched_setscheduler()* function shall set the scheduling policy and scheduling parameters of
38714 the process specified by *pid* to *policy* and the parameters specified in the **sched_param** structure
38715 pointed to by *param*, respectively. The value of the *sched_priority* member in the **sched_param**
38716 structure shall be any integer within the inclusive priority range for the scheduling policy
38717 specified by *policy*. If the value of *pid* is negative, the behavior of the *sched_setscheduler()*
38718 function is unspecified.

38719 The possible values for the *policy* parameter are defined in the **<sched.h>** header.

38720 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
38721 policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

38722 If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling
38723 process.

38724 The conditions under which one process has the appropriate privilege to change the scheduling
38725 parameters of another process are implementation-defined.

38726 Implementations may require that the requesting process have permission to set its own
38727 scheduling parameters or those of another process. Additionally, implementation-defined
38728 restrictions may apply as to the appropriate privileges required to set a process' own scheduling
38729 policy, or another process' scheduling policy, to a particular value.

38730 The *sched_setscheduler()* function shall be considered successful if it succeeds in setting the
38731 scheduling policy and scheduling parameters of the process specified by *pid* to the values
38732 specified by *policy* and the structure pointed to by *param*, respectively.

38733 ss If the scheduling policy specified by *policy* is SCHED_SPORADIC, the value specified by the
38734 *sched_ss_low_priority* member of the *param* argument shall be any integer within the inclusive
38735 priority range for the sporadic server policy. The *sched_ss_repl_period* and *sched_ss_init_budget*
38736 members of the *param* argument shall represent the time parameters used by the sporadic server
38737 scheduling policy for the target process. The *sched_ss_max_repl* member of the *param* argument
38738 shall represent the maximum number of replenishments that are allowed to be pending
38739 simultaneously for the process scheduled under this scheduling policy.

38740 The specified *sched_ss_repl_period* shall be greater than or equal to the specified
38741 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

38742 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
38743 function to succeed; if not, the function shall fail.

38744 If the scheduling policy specified by *policy* is either SCHED_FIFO or SCHED_RR, the
38745 *sched_ss_low_priority*, *sched_ss_repl_period*, and *sched_ss_init_budget* members of the *param*
38746 argument shall have no effect on the scheduling behavior.

38747 The effect of this function on individual threads is dependent on the scheduling contention
38748 scope of the threads:

38749 • For threads with system scheduling contention scope, these functions shall have no effect on
38750 their scheduling.

38751 • For threads with process scheduling contention scope, the threads' scheduling policy and
38752 associated parameters shall not be affected. However, the scheduling of these threads with
38753 respect to threads in other processes may be dependent on the scheduling parameters of their
38754 process, which are governed using these functions.

38755 If an implementation supports a two-level scheduling model in which library threads are
38756 multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled
38757 entities for the system contention scope threads shall not be affected by these functions.

38758 The underlying kernel-scheduled entities for the process contention scope threads shall have
38759 their scheduling policy and associated scheduling parameters changed to the values specified in
38760 *policy* and *param*, respectively. Kernel-scheduled entities for use by process contention scope
38761 threads that are created after this call completes shall inherit their scheduling policy and
38762 associated scheduling parameters from the process.

38763 This function is not atomic with respect to other threads in the process. Threads may continue to
38764 execute while this function call is in the process of changing the scheduling policy and
38765 associated scheduling parameters for the underlying kernel-scheduled entities used by the
38766 process contention scope threads.

38767 **RETURN VALUE**

38768 Upon successful completion, the function shall return the former scheduling policy of the
38769 specified process. If the *sched_setscheduler()* function fails to complete successfully, the policy
38770 and scheduling parameters shall remain unchanged, and the function shall return a value of *-1*
38771 and set *errno* to indicate the error.

38772 **ERRORS**

38773 The *sched_setscheduler()* function shall fail if:

38774 [EINVAL] The value of the *policy* parameter is invalid, or one or more of the parameters
38775 contained in *param* is outside the valid range for the specified scheduling
38776 policy.

38777 [EPERM] The requesting process does not have permission to set either or both of the
38778 scheduling parameters or the scheduling policy of the specified process.

38779 [ESRCH] No process can be found corresponding to that specified by *pid*.

38780 **EXAMPLES**

38781 None.

38782 **APPLICATION USAGE**

38783 None.

38784 **RATIONALE**

38785 None.

38786 **FUTURE DIRECTIONS**

38787 None.

38788 **SEE ALSO**

38789 *sched_getparam()*, *sched_getscheduler()*, *sched_setparam()*, the Base Definitions volume of
38790 IEEE Std 1003.1-2001, <**sched.h**>

38791 **CHANGE HISTORY**

38792 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38793 **Issue 6**

38794 The *sched_setscheduler()* function is marked as part of the Process Scheduling option.

38795 The [ENOSYS] error condition has been removed as stubs need not be provided if an
38796 implementation does not support the Process Scheduling option.

38797 The following new requirements on POSIX implementations derive from alignment with the
38798 Single UNIX Specification:

38799 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
38800 added.

38801 • Sections describing two-level scheduling and atomicity of the function are added.

38802 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

38803 **NAME**

38804 sched_yield — yield the processor

38805 **SYNOPSIS**38806 PS|THR `#include <sched.h>`38807 `int sched_yield(void);`

38808

38809 **DESCRIPTION**

38810 The *sched_yield()* function shall force the running thread to relinquish the processor until it again
38811 becomes the head of its thread list. It takes no arguments.

38812 **RETURN VALUE**

38813 The *sched_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a
38814 value of -1 and set *errno* to indicate the error.

38815 **ERRORS**

38816 No errors are defined.

38817 **EXAMPLES**

38818 None.

38819 **APPLICATION USAGE**

38820 None.

38821 **RATIONALE**

38822 None.

38823 **FUTURE DIRECTIONS**

38824 None.

38825 **SEE ALSO**38826 The Base Definitions volume of IEEE Std 1003.1-2001, <**sched.h**>38827 **CHANGE HISTORY**

38828 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
38829 POSIX Threads Extension.

38830 **Issue 6**38831 The *sched_yield()* function is now marked as part of the Process Scheduling and Threads options.

38832 **NAME**

38833 seed48 — seed a uniformly distributed pseudo-random non-negative long integer generator

38834 **SYNOPSIS**

38835 xSI #include <stdlib.h>

38836 unsigned short *seed48(unsigned short *seed16v*[3]);

38837

38838 **DESCRIPTION**38839 Refer to *drand48()*.

38840 NAME

38841 seekdir — set the position of a directory stream

38842 SYNOPSIS

38843 XSI `#include <dirent.h>`

38844 `void seekdir(DIR *dirp, long loc);`

38845

38846 DESCRIPTION

38847 The *seekdir()* function shall set the position of the next *readdir()* operation on the directory
38848 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been
38849 returned from an earlier call to *telldir()*. The new position reverts to the one associated with the
38850 directory stream when *telldir()* was performed.

38851 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*
38852 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to
38853 *readdir()* are unspecified.

38854 RETURN VALUE

38855 The *seekdir()* function shall not return a value.

38856 ERRORS

38857 No errors are defined.

38858 EXAMPLES

38859 None.

38860 APPLICATION USAGE

38861 None.

38862 RATIONALE

38863 The original standard developers perceived that there were restrictions on the use of the
38864 *seekdir()* and *telldir()* functions related to implementation details, and for that reason these
38865 functions need not be supported on all POSIX-conforming systems. They are required on
38866 implementations supporting the XSI extension.

38867 One of the perceived problems of implementation is that returning to a given point in a directory
38868 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-
38869 trees, hashing functions, or other similar mechanisms to order their directories are considered.
38870 The definition of *seekdir()* and *telldir()* does not specify whether, when using these interfaces, a
38871 given directory entry will be seen at all, or more than once.

38872 On systems not supporting these functions, their capability can sometimes be accomplished by
38873 saving a filename found by *readdir()* and later using *rewinddir()* and a loop on *readdir()* to
38874 relocate the position from which the filename was saved.

38875 FUTURE DIRECTIONS

38876 None.

38877 SEE ALSO

38878 *opendir()*, *readdir()*, *telldir()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**,
38879 **<stdio.h>**, **<sys/types.h>**

38880 CHANGE HISTORY

38881 First released in Issue 2.

38882 **Issue 6**

38883 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

38884 **NAME**

38885 select — synchronous I/O multiplexing

38886 **SYNOPSIS**

38887 #include <sys/time.h>

```
38888       int select(int nfds, fd_set *restrict readfds,  
38889                 fd_set *restrict writefds, fd_set *restrict errorfds,  
38890                 struct timeval *restrict timeout);
```

38891

38892 **DESCRIPTION**38893 Refer to *pselect()*.

38894 **NAME**38895 sem_close — close a named semaphore (**REALTIME**)38896 **SYNOPSIS**38897 SEM `#include <semaphore.h>`38898 `int sem_close(sem_t *sem);`

38899

38900 **DESCRIPTION**

38901 The `sem_close()` function shall indicate that the calling process is finished using the named
 38902 semaphore indicated by `sem`. The effects of calling `sem_close()` for an unnamed semaphore (one
 38903 created by `sem_init()`) are undefined. The `sem_close()` function shall deallocate (that is, make
 38904 available for reuse by a subsequent `sem_open()` by this process) any system resources allocated
 38905 by the system for use by this process for this semaphore. The effect of subsequent use of the
 38906 semaphore indicated by `sem` by this process is undefined. If the semaphore has not been
 38907 removed with a successful call to `sem_unlink()`, then `sem_close()` has no effect on the state of the
 38908 semaphore. If the `sem_unlink()` function has been successfully invoked for `name` after the most
 38909 recent call to `sem_open()` with `O_CREAT` for this semaphore, then when all processes that have
 38910 opened the semaphore close it, the semaphore is no longer accessible.

38911 **RETURN VALUE**

38912 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be
 38913 returned and `errno` set to indicate the error.

38914 **ERRORS**38915 The `sem_close()` function shall fail if:38916 [EINVAL] The `sem` argument is not a valid semaphore descriptor.38917 **EXAMPLES**

38918 None.

38919 **APPLICATION USAGE**

38920 The `sem_close()` function is part of the Semaphores option and need not be available on all
 38921 implementations.

38922 **RATIONALE**

38923 None.

38924 **FUTURE DIRECTIONS**

38925 None.

38926 **SEE ALSO**

38927 `semctl()`, `semget()`, `semop()`, `sem_init()`, `sem_open()`, `sem_unlink()`, the Base Definitions volume of
 38928 IEEE Std 1003.1-2001, **<semaphore.h>**

38929 **CHANGE HISTORY**

38930 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38931 **Issue 6**38932 The `sem_close()` function is marked as part of the Semaphores option.

38933 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 38934 implementation does not support the Semaphores option.

38935 NAME

38936 `sem_destroy` — destroy an unnamed semaphore (**REALTIME**)

38937 SYNOPSIS

38938 SEM `#include <semaphore.h>`

38939 `int sem_destroy(sem_t *sem);`

38940

38941 DESCRIPTION

38942 The `sem_destroy()` function shall destroy the unnamed semaphore indicated by `sem`. Only a
 38943 semaphore that was created using `sem_init()` may be destroyed using `sem_destroy()`; the effect of
 38944 calling `sem_destroy()` with a named semaphore is undefined. The effect of subsequent use of the
 38945 semaphore `sem` is undefined until `sem` is reinitialized by another call to `sem_init()`.

38946 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The
 38947 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

38948 RETURN VALUE

38949 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be
 38950 returned and `errno` set to indicate the error.

38951 ERRORS

38952 The `sem_destroy()` function shall fail if:

38953 [EINVAL] The `sem` argument is not a valid semaphore.

38954 The `sem_destroy()` function may fail if:

38955 [EBUSY] There are currently processes blocked on the semaphore.

38956 EXAMPLES

38957 None.

38958 APPLICATION USAGE

38959 The `sem_destroy()` function is part of the Semaphores option and need not be available on all
 38960 implementations.

38961 RATIONALE

38962 None.

38963 FUTURE DIRECTIONS

38964 None.

38965 SEE ALSO

38966 `semctl()`, `semget()`, `semop()`, `sem_init()`, `sem_open()`, the Base Definitions volume of
 38967 IEEE Std 1003.1-2001, **<semaphore.h>**

38968 CHANGE HISTORY

38969 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38970 Issue 6

38971 The `sem_destroy()` function is marked as part of the Semaphores option.

38972 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 38973 implementation does not support the Semaphores option.

38974 **NAME**38975 `sem_getvalue` — get the value of a semaphore (**REALTIME**)38976 **SYNOPSIS**38977 SEM `#include <semaphore.h>`38978 `int sem_getvalue(sem_t *restrict sem, int *restrict sval);`

38979

38980 **DESCRIPTION**

38981 The `sem_getvalue()` function shall update the location referenced by the `sval` argument to have
 38982 the value of the semaphore referenced by `sem` without affecting the state of the semaphore. The
 38983 updated value represents an actual semaphore value that occurred at some unspecified time
 38984 during the call, but it need not be the actual value of the semaphore when it is returned to the
 38985 calling process.

38986 If `sem` is locked, then the object to which `sval` points shall either be set to zero or to a negative
 38987 number whose absolute value represents the number of processes waiting for the semaphore at
 38988 some unspecified time during the call.

38989 **RETURN VALUE**

38990 Upon successful completion, the `sem_getvalue()` function shall return a value of zero. Otherwise,
 38991 it shall return a value of `-1` and set `errno` to indicate the error.

38992 **ERRORS**38993 The `sem_getvalue()` function shall fail if:38994 [EINVAL] The `sem` argument does not refer to a valid semaphore.38995 **EXAMPLES**

38996 None.

38997 **APPLICATION USAGE**

38998 The `sem_getvalue()` function is part of the Semaphores option and need not be available on all
 38999 implementations.

39000 **RATIONALE**

39001 None.

39002 **FUTURE DIRECTIONS**

39003 None.

39004 **SEE ALSO**

39005 `semctl()`, `semget()`, `semop()`, `sem_post()`, `sem_timedwait()`, `sem_trywait()`, `sem_wait()`, the Base
 39006 Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**

39007 **CHANGE HISTORY**

39008 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39009 **Issue 6**39010 The `sem_getvalue()` function is marked as part of the Semaphores option.

39011 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 39012 implementation does not support the Semaphores option.

39013 The `sem_timedwait()` function is added to the SEE ALSO section for alignment with
 39014 IEEE Std 1003.1d-1999.

39015 The **restrict** keyword is added to the `sem_getvalue()` prototype for alignment with the
 39016 ISO/IEC 9899:1999 standard.

39017

IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/54 is applied.

39018 **NAME**39019 sem_init — initialize an unnamed semaphore (**REALTIME**)39020 **SYNOPSIS**39021 SEM `#include <semaphore.h>`39022 `int sem_init(sem_t *sem, int pshared, unsigned value);`

39023

39024 **DESCRIPTION**

39025 The *sem_init()* function shall initialize the unnamed semaphore referred to by *sem*. The value of
 39026 the initialized semaphore shall be *value*. Following a successful call to *sem_init()*, the semaphore
 39027 may be used in subsequent calls to *sem_wait()*, *sem_trywait()*, *sem_post()*, and *sem_destroy()*.
 39028 This semaphore shall remain usable until the semaphore is destroyed.

39029 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;
 39030 in this case, any process that can access the semaphore *sem* can use *sem* for performing
 39031 *sem_wait()*, *sem_trywait()*, *sem_post()*, and *sem_destroy()* operations.

39032 Only *sem* itself may be used for performing synchronization. The result of referring to copies of
 39033 *sem* in calls to *sem_wait()*, *sem_trywait()*, *sem_post()*, and *sem_destroy()* is undefined.

39034 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any
 39035 thread in this process can use *sem* for performing *sem_wait()*, *sem_trywait()*, *sem_post()*, and
 39036 *sem_destroy()* operations. The use of the semaphore by threads other than those created in the
 39037 same process is undefined.

39038 Attempting to initialize an already initialized semaphore results in undefined behavior.

39039 **RETURN VALUE**

39040 Upon successful completion, the *sem_init()* function shall initialize the semaphore in *sem*.
 39041 Otherwise, it shall return `-1` and set *errno* to indicate the error.

39042 **ERRORS**39043 The *sem_init()* function shall fail if:

39044	[EINVAL]	The <i>value</i> argument exceeds {SEM_VALUE_MAX}.
39045	[ENOSPC]	A resource required to initialize the semaphore has been exhausted, or the
39046		limit on semaphores ({SEM_NSEMS_MAX}) has been reached.
39047	[EPERM]	The process lacks the appropriate privileges to initialize the semaphore.

39048 **EXAMPLES**

39049 None.

39050 **APPLICATION USAGE**

39051 The *sem_init()* function is part of the Semaphores option and need not be available on all
 39052 implementations.

39053 **RATIONALE**

39054 Although this volume of IEEE Std 1003.1-2001 fails to specify a successful return value, it is
 39055 likely that a later version may require the implementation to return a value of zero if the call to
 39056 *sem_init()* is successful.

39057 **FUTURE DIRECTIONS**

39058 None.

39059 **SEE ALSO**

39060 *sem_destroy()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_wait()*, the Base Definitions
39061 volume of IEEE Std 1003.1-2001, <**semaphore.h**>

39062 **CHANGE HISTORY**

39063 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39064 **Issue 6**

39065 The *sem_init()* function is marked as part of the Semaphores option.

39066 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39067 implementation does not support the Semaphores option.

39068 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with
39069 IEEE Std 1003.1d-1999.

39070 **NAME**39071 sem_open — initialize and open a named semaphore (**REALTIME**)39072 **SYNOPSIS**39073 SEM `#include <semaphore.h>`39074 `sem_t *sem_open(const char *name, int oflag, ...);`

39075

39076 **DESCRIPTION**

39077 The *sem_open()* function shall establish a connection between a named semaphore and a process.
 39078 Following a call to *sem_open()* with semaphore name *name*, the process may reference the
 39079 semaphore associated with *name* using the address returned from the call. This semaphore may
 39080 be used in subsequent calls to *sem_wait()*, *sem_trywait()*, *sem_post()*, and *sem_close()*. The
 39081 semaphore remains usable by this process until the semaphore is closed by a successful call to
 39082 *sem_close()*, *_exit()*, or one of the *exec* functions.

39083 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to
 39084 *sem_open()*. The following flag bits may be set in *oflag*:

39085 **O_CREAT** This flag is used to create a semaphore if it does not already exist. If **O_CREAT** is
 39086 set and the semaphore already exists, then **O_CREAT** has no effect, except as noted
 39087 under **O_EXCL**. Otherwise, *sem_open()* creates a named semaphore. The **O_CREAT**
 39088 flag requires a third and a fourth argument: *mode*, which is of type **mode_t**, and
 39089 *value*, which is of type **unsigned**. The semaphore is created with an initial value of
 39090 *value*. Valid initial values for semaphores are less than or equal to
 39091 {SEM_VALUE_MAX}.

39092 The user ID of the semaphore is set to the effective user ID of the process; the
 39093 group ID of the semaphore is set to a system default group ID or to the effective
 39094 group ID of the process. The permission bits of the semaphore are set to the value
 39095 of the *mode* argument except those set in the file mode creation mask of the
 39096 process. When bits in *mode* other than the file permission bits are specified, the
 39097 effect is unspecified.

39098 After the semaphore named *name* has been created by *sem_open()* with the
 39099 **O_CREAT** flag, other processes can connect to the semaphore by calling
 39100 *sem_open()* with the same value of *name*.

39101 **O_EXCL** If **O_EXCL** and **O_CREAT** are set, *sem_open()* fails if the semaphore *name* exists.
 39102 The check for the existence of the semaphore and the creation of the semaphore if
 39103 it does not exist are atomic with respect to other processes executing *sem_open()*
 39104 with **O_EXCL** and **O_CREAT** set. If **O_EXCL** is set and **O_CREAT** is not set, the
 39105 effect is undefined.

39106 If flags other than **O_CREAT** and **O_EXCL** are specified in the *oflag* parameter, the
 39107 effect is unspecified.

39108 The *name* argument points to a string naming a semaphore object. It is unspecified whether the
 39109 name appears in the file system and is visible to functions that take pathnames as arguments.
 39110 The *name* argument conforms to the construction rules for a pathname. If *name* begins with the
 39111 slash character, then processes calling *sem_open()* with the same value of *name* shall refer to the
 39112 same semaphore object, as long as that name has not been removed. If *name* does not begin with
 39113 the slash character, the effect is implementation-defined. The interpretation of slash characters
 39114 other than the leading slash character in *name* is implementation-defined.

39115 If a process makes multiple successful calls to *sem_open()* with the same value for *name*, the
 39116 same semaphore address shall be returned for each such successful call, provided that there

39117 have been no calls to *sem_unlink()* for this semaphore.

39118 References to copies of the semaphore produce undefined results.

39119 RETURN VALUE

39120 Upon successful completion, the *sem_open()* function shall return the address of the semaphore.

39121 Otherwise, it shall return a value of SEM_FAILED and set *errno* to indicate the error. The symbol

39122 SEM_FAILED is defined in the **<semaphore.h>** header. No successful return from *sem_open()*

39123 shall return the value SEM_FAILED.

39124 ERRORS

39125 If any of the following conditions occur, the *sem_open()* function shall return SEM_FAILED and
39126 set *errno* to the corresponding value:

39127 [EACCES] The named semaphore exists and the permissions specified by *oflag* are
39128 denied, or the named semaphore does not exist and permission to create the
39129 named semaphore is denied.

39130 [EEXIST] O_CREAT and O_EXCL are set and the named semaphore already exists.

39131 [EINTR] The *sem_open()* operation was interrupted by a signal.

39132 [EINVAL] The *sem_open()* operation is not supported for the given name, or O_CREAT
39133 was specified in *oflag* and *value* was greater than {SEM_VALUE_MAX}.

39134 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by
39135 this process.

39136 [ENAMETOOLONG]

39137 The length of the *name* argument exceeds {PATH_MAX} or a pathname
39138 component is longer than {NAME_MAX}.

39139 [ENFILE] Too many semaphores are currently open in the system.

39140 [ENOENT] O_CREAT is not set and the named semaphore does not exist.

39141 [ENOSPC] There is insufficient space for the creation of the new named semaphore.

39142 EXAMPLES

39143 None.

39144 APPLICATION USAGE

39145 The *sem_open()* function is part of the Semaphores option and need not be available on all
39146 implementations.

39147 RATIONALE

39148 Early drafts required an error return value of -1 with the type **sem_t *** for the *sem_open()*
39149 function, which is not guaranteed to be portable across implementations. The revised text
39150 provides the symbolic error code SEM_FAILED to eliminate the type conflict.

39151 FUTURE DIRECTIONS

39152 None.

39153 SEE ALSO

39154 *semctl()*, *semget()*, *semop()*, *sem_close()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*,

39155 *sem_wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<semaphore.h>**

39156 CHANGE HISTORY

39157 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39158 **Issue 6**

39159 The *sem_open()* function is marked as part of the Semaphores option.

39160 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39161 implementation does not support the Semaphores option.

39162 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with
39163 IEEE Std 1003.1d-1999.

39164 **NAME**39165 sem_post — unlock a semaphore (**REALTIME**)39166 **SYNOPSIS**39167 SEM `#include <semaphore.h>`39168 `int sem_post(sem_t *sem);`

39169

39170 **DESCRIPTION**39171 The `sem_post()` function shall unlock the semaphore referenced by `sem` by performing a
39172 semaphore unlock operation on that semaphore.39173 If the semaphore value resulting from this operation is positive, then no threads were blocked
39174 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.39175 If the value of the semaphore resulting from this operation is zero, then one of the threads
39176 blocked waiting for the semaphore shall be allowed to return successfully from its call to
39177 PS `sem_wait()`. If the Process Scheduling option is supported, the thread to be unblocked shall be
39178 chosen in a manner appropriate to the scheduling policies and parameters in effect for the
39179 blocked threads. In the case of the schedulers `SCHED_FIFO` and `SCHED_RR`, the highest
39180 priority waiting thread shall be unblocked, and if there is more than one highest priority thread
39181 blocked waiting for the semaphore, then the highest priority thread that has been waiting the
39182 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread
39183 to unblock is unspecified.39184 SS If the Process Sporadic Server option is supported, and the scheduling policy is
39185 `SCHED_SPORADIC`, the semantics are as per `SCHED_FIFO` above.39186 The `sem_post()` function shall be reentrant with respect to signals and may be invoked from a
39187 signal-catching function.39188 **RETURN VALUE**39189 If successful, the `sem_post()` function shall return zero; otherwise, the function shall return `-1`
39190 and set `errno` to indicate the error.39191 **ERRORS**39192 The `sem_post()` function shall fail if:39193 [EINVAL] The `sem` argument does not refer to a valid semaphore.39194 **EXAMPLES**

39195 None.

39196 **APPLICATION USAGE**39197 The `sem_post()` function is part of the Semaphores option and need not be available on all
39198 implementations.39199 **RATIONALE**

39200 None.

39201 **FUTURE DIRECTIONS**

39202 None.

39203 **SEE ALSO**39204 `semctl()`, `semget()`, `semop()`, `sem_timedwait()`, `sem_trywait()`, `sem_wait()`, the Base Definitions
39205 volume of IEEE Std 1003.1-2001, `<semaphore.h>`

CHANGE HISTORY

39207 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

39209 The *sem_post()* function is marked as part of the Semaphores option.

39210 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39211 implementation does not support the Semaphores option.

39212 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with
39213 IEEE Std 1003.1d-1999.

39214 SCHED_SPORADIC is added to the list of scheduling policies for which the thread that is to be
39215 unblocked is specified for alignment with IEEE Std 1003.1d-1999.

39216 **NAME**39217 `sem_timedwait` — lock a semaphore (**ADVANCED REALTIME**)39218 **SYNOPSIS**39219 SEM TMO `#include <semaphore.h>`39220 `#include <time.h>`

```
39221 int sem_timedwait(sem_t *restrict sem,
39222                  const struct timespec *restrict abs_timeout);
39223
```

39224 **DESCRIPTION**

39225 The `sem_timedwait()` function shall lock the semaphore referenced by `sem` as in the `sem_wait()`
 39226 function. However, if the semaphore cannot be locked without waiting for another process or
 39227 thread to unlock the semaphore by performing a `sem_post()` function, this wait shall be
 39228 terminated when the specified timeout expires.

39229 The timeout shall expire when the absolute time specified by `abs_timeout` passes, as measured by
 39230 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 39231 `abs_timeout`), or if the absolute time specified by `abs_timeout` has already been passed at the time
 39232 of the call.

39233 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If
 39234 the Timers option is not supported, the timeout shall be based on the system clock as returned
 39235 by the `time()` function. The resolution of the timeout shall be the resolution of the clock on which
 39236 it is based. The **timespec** data type is defined as a structure in the `<time.h>` header.

39237 Under no circumstance shall the function fail with a timeout if the semaphore can be locked
 39238 immediately. The validity of the `abs_timeout` need not be checked if the semaphore can be locked
 39239 immediately.

39240 **RETURN VALUE**

39241 The `sem_timedwait()` function shall return zero if the calling process successfully performed the
 39242 semaphore lock operation on the semaphore designated by `sem`. If the call was unsuccessful, the
 39243 state of the semaphore shall be unchanged, and the function shall return a value of `-1` and set
 39244 `errno` to indicate the error.

39245 **ERRORS**39246 The `sem_timedwait()` function shall fail if:

39247 [EINVAL] The `sem` argument does not refer to a valid semaphore.

39248 [EINVAL] The process or thread would have blocked, and the `abs_timeout` parameter
 39249 specified a nanoseconds field value less than zero or greater than or equal to
 39250 1 000 million.

39251 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

39252 The `sem_timedwait()` function may fail if:

39253 [EDEADLK] A deadlock condition was detected.

39254 [EINTR] A signal interrupted this function.

39255 EXAMPLES

39256 None.

39257 APPLICATION USAGE

39258 Applications using these functions may be subject to priority inversion, as discussed in the Base
39259 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

39260 The *sem_timedwait()* function is part of the Semaphores and Timeouts options and need not be
39261 provided on all implementations.

39262 RATIONALE

39263 None.

39264 FUTURE DIRECTIONS

39265 None.

39266 SEE ALSO

39267 *sem_post()*, *sem_trywait()*, *sem_wait()*, *semctl()*, *semget()*, *semop()*, *time()*, the Base Definitions
39268 volume of IEEE Std 1003.1-2001, <**semaphore.h**>, <**time.h**>

39269 CHANGE HISTORY

39270 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

39271 **NAME**39272 sem_trywait, sem_wait — lock a semaphore (**REALTIME**)39273 **SYNOPSIS**39274 SEM `#include <semaphore.h>`39275 `int sem_trywait(sem_t *sem);`39276 `int sem_wait(sem_t *sem);`

39277

39278 **DESCRIPTION**

39279 The `sem_trywait()` function shall lock the semaphore referenced by `sem` only if the semaphore is
 39280 currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not
 39281 lock the semaphore.

39282 The `sem_wait()` function shall lock the semaphore referenced by `sem` by performing a semaphore
 39283 lock operation on that semaphore. If the semaphore value is currently zero, then the calling
 39284 thread shall not return from the call to `sem_wait()` until it either locks the semaphore or the call is
 39285 interrupted by a signal.

39286 Upon successful return, the state of the semaphore shall be locked and shall remain locked until
 39287 the `sem_post()` function is executed and returns successfully.

39288 The `sem_wait()` function is interruptible by the delivery of a signal.

39289 **RETURN VALUE**

39290 The `sem_trywait()` and `sem_wait()` functions shall return zero if the calling process successfully
 39291 performed the semaphore lock operation on the semaphore designated by `sem`. If the call was
 39292 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a
 39293 value of `-1` and set `errno` to indicate the error.

39294 **ERRORS**

39295 The `sem_trywait()` and `sem_wait()` functions shall fail if:

39296 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the
 39297 `sem_trywait()` operation (`sem_trywait()` only).

39298 [EINVAL] The `sem` argument does not refer to a valid semaphore.

39299 The `sem_trywait()` and `sem_wait()` functions may fail if:

39300 [EDEADLK] A deadlock condition was detected.

39301 [EINTR] A signal interrupted this function.

39302 **EXAMPLES**

39303 None.

39304 **APPLICATION USAGE**

39305 Applications using these functions may be subject to priority inversion, as discussed in the Base
 39306 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

39307 The `sem_trywait()` and `sem_wait()` functions are part of the Semaphores option and need not be
 39308 provided on all implementations.

39309 **RATIONALE**

39310 None.

39311 **FUTURE DIRECTIONS**

39312 None.

39313 **SEE ALSO**

39314 *semctl()*, *semget()*, *semop()*, *sem_post()*, *sem_timedwait()*, the Base Definitions volume of
39315 IEEE Std 1003.1-2001, <**semaphore.h**>

39316 **CHANGE HISTORY**

39317 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39318 **Issue 6**39319 The *sem_trywait()* and *sem_wait()* functions are marked as part of the Semaphores option.

39320 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39321 implementation does not support the Semaphores option.

39322 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with
39323 IEEE Std 1003.1d-1999.

39324 **NAME**39325 sem_unlink — remove a named semaphore (**REALTIME**)39326 **SYNOPSIS**39327 SEM `#include <semaphore.h>`39328 `int sem_unlink(const char *name);`

39329

39330 **DESCRIPTION**

39331 The `sem_unlink()` function shall remove the semaphore named by the string *name*. If the
 39332 semaphore named by *name* is currently referenced by other processes, then `sem_unlink()` shall
 39333 have no effect on the state of the semaphore. If one or more processes have the semaphore open
 39334 when `sem_unlink()` is called, destruction of the semaphore is postponed until all references to the
 39335 semaphore have been destroyed by calls to `sem_close()`, `_exit()`, or `exec`. Calls to `sem_open()` to
 39336 recreate or reconnect to the semaphore refer to a new semaphore after `sem_unlink()` is called. The
 39337 `sem_unlink()` call shall not block until all references have been destroyed; it shall return
 39338 immediately.

39339 **RETURN VALUE**

39340 Upon successful completion, the `sem_unlink()` function shall return a value of 0. Otherwise, the
 39341 semaphore shall not be changed and the function shall return a value of `-1` and set *errno* to
 39342 indicate the error.

39343 **ERRORS**39344 The `sem_unlink()` function shall fail if:

39345 [EACCES] Permission is denied to unlink the named semaphore.

39346 [ENAMETOOLONG]

39347 The length of the *name* argument exceeds {PATH_MAX} or a pathname
 39348 component is longer than {NAME_MAX}.

39349 [ENOENT] The named semaphore does not exist.

39350 **EXAMPLES**

39351 None.

39352 **APPLICATION USAGE**

39353 The `sem_unlink()` function is part of the Semaphores option and need not be available on all
 39354 implementations.

39355 **RATIONALE**

39356 None.

39357 **FUTURE DIRECTIONS**

39358 None.

39359 **SEE ALSO**

39360 `semctl()`, `semget()`, `semop()`, `sem_close()`, `sem_open()`, the Base Definitions volume of
 39361 IEEE Std 1003.1-2001, <**semaphore.h**>

39362 **CHANGE HISTORY**

39363 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39364 **Issue 6**39365 The `sem_unlink()` function is marked as part of the Semaphores option.

39366 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39367 implementation does not support the Semaphores option.

39368 **NAME**

39369 `sem_wait` — lock a semaphore (**REALTIME**)

39370 **SYNOPSIS**

39371 SEM `#include <semaphore.h>`

39372 `int sem_wait(sem_t *sem);`

39373

39374 **DESCRIPTION**

39375 Refer to `sem_trywait()`.

39376 **NAME**

39377 semctl — XSI semaphore control operations

39378 **SYNOPSIS**39379 xsi `#include <sys/sem.h>`39380 `int semctl(int semid, int semnum, int cmd, ...);`

39381

39382 **DESCRIPTION**

39383 The *semctl()* function operates on XSI semaphores (see the Base Definitions volume of
 39384 IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function
 39385 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on
 39386 page 41).

39387 The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.
 39388 The fourth argument is optional and depends upon the operation requested. If required, it is of
 39389 type **union semun**, which the application shall explicitly declare:

```
39390 union semun {
39391     int val;
39392     struct semid_ds *buf;
39393     unsigned short *array;
39394 } arg;
```

39395 The following semaphore control operations as specified by *cmd* are executed with respect to the
 39396 semaphore specified by *semid* and *semnum*. The level of permission required for each operation
 39397 is shown with each command; see Section 2.7 (on page 39). The symbolic names for the values
 39398 of *cmd* are defined in the **<sys/sem.h>** header:

39399	GETVAL	Return the value of <i>semval</i> ; see <sys/sem.h> . Requires read permission.
39400	SETVAL	Set the value of <i>semval</i> to <i>arg.val</i> , where <i>arg</i> is the value of the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> value corresponding to the specified semaphore in all processes is cleared. Requires alter permission; see Section 2.7 (on page 39).
39401		
39402		
39403		
39404	GETPID	Return the value of <i>sempid</i> . Requires read permission.
39405	GETNCNT	Return the value of <i>semncnt</i> . Requires read permission.
39406	GETZCNT	Return the value of <i>semzcnt</i> . Requires read permission.

39407 The following values of *cmd* operate on each *semval* in the set of semaphores:

39408	GETALL	Return the value of <i>semval</i> for each semaphore in the semaphore set and place into the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . Requires read permission.
39409		
39410		
39411	SETALL	Set the value of <i>semval</i> for each semaphore in the semaphore set according to the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> values corresponding to each specified semaphore in all processes are cleared. Requires alter permission.
39412		
39413		
39414		
39415		

39416 The following values of *cmd* are also available:

39417	IPC_STAT	Place the current value of each member of the semid_ds data structure associated with <i>semid</i> into the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . The contents of this structure are defined in
39418		
39419		

39420		<sys/sem.h>. Requires read permission.
39421	IPC_SET	Set the value of the following members of the semid_ds data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> :
39422		
39423		
39424		<i>sem_perm.uid</i>
39425		<i>sem_perm.gid</i>
39426		<i>sem_perm.mode</i>
39427		The mode bits specified in Section 2.7.1 (on page 40) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i> . The stored values of any other bits are unspecified.
39428		
39429		
39430		This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i> .
39431		
39432		
39433		
39434	IPC_RMID	Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and semid_ds data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i> .
39435		
39436		
39437		
39438		
39439		
39440	RETURN VALUE	
39441		If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:
39442	GETVAL	The value of <i>semval</i> .
39443	GETPID	The value of <i>sempid</i> .
39444	GETNCNT	The value of <i>semncnt</i> .
39445	GETZCNT	The value of <i>semzcnt</i> .
39446	All others	0.
39447		Otherwise, <i>semctl()</i> shall return -1 and set <i>errno</i> to indicate the error.
39448	ERRORS	
39449		The <i>semctl()</i> function shall fail if:
39450	[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 39).
39451		
39452	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i> , or the value of <i>cmd</i> is not a valid command.
39453		
39454		
39455	[EPERM]	The argument <i>cmd</i> is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the data structure associated with <i>semid</i> .
39456		
39457		
39458		
39459	[ERANGE]	The argument <i>cmd</i> is equal to SETVAL or SETALL and the value to which <i>semval</i> is to be set is greater than the system-imposed maximum.
39460		

39461 **EXAMPLES**

39462 None.

39463 **APPLICATION USAGE**

39464 The fourth parameter in the SYNOPSIS section is now specified as ". . ." in order to avoid a
39465 clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for
39466 backwards-compatibility.

39467 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
39468 Application developers who need to use IPC should design their applications so that modules
39469 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the
39470 alternative interfaces.

39471 **RATIONALE**

39472 None.

39473 **FUTURE DIRECTIONS**

39474 None.

39475 **SEE ALSO**

39476 Section 2.7 (on page 39), Section 2.8 (on page 41), *semget()*, *semop()*, *sem_close()*, *sem_destroy()*,
39477 *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_unlink()*, *sem_wait()*, the Base Definitions
39478 volume of IEEE Std 1003.1-2001, <**sys/sem.h**>

39479 **CHANGE HISTORY**

39480 First released in Issue 2. Derived from Issue 2 of the SVID.

39481 **Issue 5**

39482 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
39483 DIRECTIONS to the APPLICATION USAGE section.

39484 **NAME**

39485 semget — get set of XSI semaphores

39486 **SYNOPSIS**

39487 XSI #include <sys/sem.h>

39488 int semget(key_t key, int nsems, int semflg);

39489

39490 **DESCRIPTION**

39491 The *semget()* function operates on XSI semaphores (see the Base Definitions volume of
 39492 IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function
 39493 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on
 39494 page 41).

39495 The *semget()* function shall return the semaphore identifier associated with *key*.

39496 A semaphore identifier with its associated **semid_ds** data structure and its associated set of
 39497 *nsems* semaphores (see <sys/sem.h>) is created for *key* if one of the following is true:

- 39498 • The argument *key* is equal to IPC_PRIVATE.
- 39499 • The argument *key* does not already have a semaphore identifier associated with it and (*semflg*
 39500 &IPC_CREAT) is non-zero.

39501 Upon creation, the **semid_ds** data structure associated with the new semaphore identifier is
 39502 initialized as follows:

- 39503 • In the operation permissions structure *sem_perm.cuid*, *sem_perm.uid*, *sem_perm.cgid*, and
 39504 *sem_perm.gid* shall be set equal to the effective user ID and effective group ID, respectively, of
 39505 the calling process.
- 39506 • The low-order 9 bits of *sem_perm.mode* shall be set equal to the low-order 9 bits of *semflg*.
- 39507 • The variable *sem_nsems* shall be set equal to the value of *nsems*.
- 39508 • The variable *sem_otime* shall be set equal to 0 and *sem_ctime* shall be set equal to the current
 39509 time.
- 39510 • The data structure associated with each semaphore in the set shall not be initialized. The
 39511 *semctl()* function with the command SETVAL or SETALL can be used to initialize each
 39512 semaphore.

39513 **RETURN VALUE**

39514 Upon successful completion, *semget()* shall return a non-negative integer, namely a semaphore
 39515 identifier; otherwise, it shall return -1 and set *errno* to indicate the error.

39516 **ERRORS**

39517 The *semget()* function shall fail if:

39518 39519 39520	[EACCES]	A semaphore identifier exists for <i>key</i> , but operation permission as specified by the low-order 9 bits of <i>semflg</i> would not be granted; see Section 2.7 (on page 39).
39521 39522	[EEXIST]	A semaphore identifier exists for the argument <i>key</i> but ((<i>semflg</i> &IPC_CREAT) &&(semflg &IPC_EXCL)) is non-zero.
39523 39524 39525 39526	[EINVAL]	The value of <i>nsems</i> is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument <i>key</i> , but the number of semaphores in the set associated with it is less than <i>nsems</i> and <i>nsems</i> is not equal to 0.

39527 [ENOENT] A semaphore identifier does not exist for the argument *key* and (*semflg*
 39528 &IPC_CREAT) is equal to 0.

39529 [ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the
 39530 maximum number of allowed semaphores system-wide would be exceeded.

39531 EXAMPLES

39532 Creating a Semaphore Identifier

39533 The following example gets a unique semaphore key using the *ftok()* function, then gets a
 39534 semaphore ID associated with that key using the *semget()* function (the first call also tests to
 39535 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as
 39536 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the
 39537 program attempts to create one semaphore with read/write permission for all. It also uses the
 39538 IPC_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

39539 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in
 39540 the *sbuf* array. The number of processes that can execute concurrently without queuing is
 39541 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in
 39542 the program.

```

39543 #include <sys/types.h>
39544 #include <stdio.h>
39545 #include <sys/ipc.h>
39546 #include <sys/sem.h>
39547 #include <sys/stat.h>
39548 #include <errno.h>
39549 #include <unistd.h>
39550 #include <stdlib.h>
39551 #include <pwd.h>
39552 #include <fcntl.h>
39553 #include <limits.h>
39554 ...
39555 key_t semkey;
39556 int semid, pfd, fv;
39557 struct sembuf sbuf;
39558 char *lgn;
39559 char filename[PATH_MAX+1];
39560 struct stat outstat;
39561 struct passwd *pw;
39562 ...
39563 /* Get unique key for semaphore. */
39564 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39565     perror("IPC error: ftok"); exit(1);
39566 }
39567
39568 /* Get semaphore ID associated with this key. */
39569 if ((semid = semget(semkey, 0, 0)) == -1) {
39570     /* Semaphore does not exist - Create. */
39571     if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39572         S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39573     {
39574         /* Initialize the semaphore. */
39575         sbuf.sem_num = 0;
```

```

39575         sbuf.sem_op = 2;  /* This is the number of runs
39576                        without queuing. */
39577         sbuf.sem_flg = 0;
39578         if (semop(semid, &sbuf, 1) == -1) {
39579             perror("IPC error: semop"); exit(1);
39580         }
39581     }
39582     else if (errno == EEXIST) {
39583         if ((semid = semget(semkey, 0, 0)) == -1) {
39584             perror("IPC error 1: semget"); exit(1);
39585         }
39586     }
39587     else {
39588         perror("IPC error 2: semget"); exit(1);
39589     }
39590 }
39591 ...

```

39592 APPLICATION USAGE

39593 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
 39594 Application developers who need to use IPC should design their applications so that modules
 39595 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the
 39596 alternative interfaces.

39597 RATIONALE

39598 None.

39599 FUTURE DIRECTIONS

39600 None.

39601 SEE ALSO

39602 Section 2.7 (on page 39), Section 2.8 (on page 41), *semctl()*, *semop()*, *sem_close()*, *sem_destroy()*,
 39603 *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_unlink()*, *sem_wait()*, the Base Definitions
 39604 volume of IEEE Std 1003.1-2001, <sys/sem.h>

39605 CHANGE HISTORY

39606 First released in Issue 2. Derived from Issue 2 of the SVID.

39607 Issue 5

39608 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 39609 DIRECTIONS to a new APPLICATION USAGE section.

39610 **NAME**

39611 semop — XSI semaphore operations

39612 **SYNOPSIS**39613 XSI `#include <sys/sem.h>`39614 `int semop(int semid, struct sembuf *sops, size_t nsops);`

39615

39616 **DESCRIPTION**

39617 The *semop()* function operates on XSI semaphores (see the Base Definitions volume of
 39618 IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function
 39619 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on
 39620 page 41).

39621 The *semop()* function shall perform atomically a user-defined array of semaphore operations on
 39622 the set of semaphores associated with the semaphore identifier specified by the argument *semid*.

39623 The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The
 39624 implementation shall not modify elements of this array unless the application uses
 39625 implementation-defined extensions.

39626 The argument *nsops* is the number of such structures in the array.

39627 Each structure, **sembuf**, includes the following members:

39628

39629

39630

39631

39632

Member Type	Member Name	Description
short	<i>sem_num</i>	Semaphore number.
short	<i>sem_op</i>	Semaphore operation.
short	<i>sem_flg</i>	Operation flags.

39633 Each semaphore operation specified by *sem_op* is performed on the corresponding semaphore
 39634 specified by *semid* and *sem_num*.

39635 The variable *sem_op* specifies one of three semaphore operations:

- 39636 1. If *sem_op* is a negative integer and the calling process has alter permission, one of the
 39637 following shall occur:
 - 39638 • If *semval*(see <sys/sem.h>) is greater than or equal to the absolute value of *sem_op*, the
 39639 absolute value of *sem_op* is subtracted from *semval*. Also, if (*sem_flg* & SEM_UNDO) is
 39640 non-zero, the absolute value of *sem_op* shall be added to the calling process' *semadj*
 39641 value for the specified semaphore.
 - 39642 • If *semval* is less than the absolute value of *sem_op* and (*sem_flg* & IPC_NOWAIT) is non-
 39643 zero, *semop()* shall return immediately.
 - 39644 • If *semval* is less than the absolute value of *sem_op* and (*sem_flg* & IPC_NOWAIT) is 0,
 39645 *semop()* shall increment the *semncnt* associated with the specified semaphore and
 39646 suspend execution of the calling thread until one of the following conditions occurs:
 - 39647 — The value of *semval* becomes greater than or equal to the absolute value of *sem_op*.
 39648 When this occurs, the value of *semncnt* associated with the specified semaphore
 39649 shall be decremented, the absolute value of *sem_op* shall be subtracted from *semval*
 39650 and, if (*sem_flg* & SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be
 39651 added to the calling process' *semadj* value for the specified semaphore.
 - 39652 — The *semid* for which the calling thread is awaiting action is removed from the
 39653 system. When this occurs, *errno* shall be set equal to [EIDRM] and *-1* shall be

39654 returned.

39655 — The calling thread receives a signal that is to be caught. When this occurs, the value
39656 of *semcnt* associated with the specified semaphore shall be decremented, and the
39657 calling thread shall resume execution in the manner prescribed in *sigaction()*.

39658 2. If *sem_op* is a positive integer and the calling process has alter permission, the value of
39659 *sem_op* shall be added to *semval* and, if (*sem_flg* & SEM_UNDO) is non-zero, the value of
39660 *sem_op* shall be subtracted from the calling process' *semadj* value for the specified
39661 semaphore.

39662 3. If *sem_op* is 0 and the calling process has read permission, one of the following shall occur:

39663 • If *semval* is 0, *semop()* shall return immediately.

39664 • If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is non-zero, *semop()* shall return
39665 immediately.

39666 • If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is 0, *semop()* shall increment the
39667 *semzcnt* associated with the specified semaphore and suspend execution of the calling
39668 thread until one of the following occurs:

39669 — The value of *semval* becomes 0, at which time the value of *semzcnt* associated with
39670 the specified semaphore shall be decremented.

39671 — The *semid* for which the calling thread is awaiting action is removed from the
39672 system. When this occurs, *errno* shall be set equal to [EIDRM] and -1 shall be
39673 returned.

39674 — The calling thread receives a signal that is to be caught. When this occurs, the value
39675 of *semzcnt* associated with the specified semaphore shall be decremented, and the
39676 calling thread shall resume execution in the manner prescribed in *sigaction()*.

39677 Upon successful completion, the value of *sempid* for each semaphore specified in the array
39678 pointed to by *sops* shall be set equal to the process ID of the calling process.

39679 **RETURN VALUE**

39680 Upon successful completion, *semop()* shall return 0; otherwise, it shall return -1 and set *errno* to
39681 indicate the error.

39682 **ERRORS**

39683 The *semop()* function shall fail if:

39684 [E2BIG] The value of *nsops* is greater than the system-imposed maximum.

39685 [EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page
39686 39).

39687 [EAGAIN] The operation would result in suspension of the calling process but (*sem_flg*
39688 & IPC_NOWAIT) is non-zero.

39689 [EFBIG] The value of *sem_num* is less than 0 or greater than or equal to the number of
39690 semaphores in the set associated with *semid*.

39691 [EIDRM] The semaphore identifier *semid* is removed from the system.

39692 [EINTR] The *semop()* function was interrupted by a signal.

39693 [EINVAL] The value of *semid* is not a valid semaphore identifier, or the number of
39694 individual semaphores for which the calling process requests a SEM_UNDO
39695 would exceed the system-imposed limit.

39696 [ENOSPC] The limit on the number of individual processes requesting a SEM_UNDO
 39697 would be exceeded.

39698 [ERANGE] An operation would cause a *semval* to overflow the system-imposed limit, or
 39699 an operation would cause a *semadj* value to overflow the system-imposed
 39700 limit.

39701 EXAMPLES

39702 Setting Values in Semaphores

39703 The following example sets the values of the two semaphores associated with the *semid*
 39704 identifier to the values contained in the *sb* array.

```
39705 #include <sys/sem.h>
39706 ...
39707 int semid;
39708 struct sembuf sb[2];
39709 int nsops = 2;
39710 int result;

39711 /* Adjust value of semaphore in the semaphore array semid. */
39712 sb[0].sem_num = 0;
39713 sb[0].sem_op = -1;
39714 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
39715 sb[1].sem_num = 1;
39716 sb[1].sem_op = 1;
39717 sb[1].sem_flg = 0;

39718 result = semop(semid, sb, nsops);
```

39719 Creating a Semaphore Identifier

39720 The following example gets a unique semaphore key using the *ftok()* function, then gets a
 39721 semaphore ID associated with that key using the *semget()* function (the first call also tests to
 39722 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as
 39723 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the
 39724 program attempts to create one semaphore with read/write permission for all. It also uses the
 39725 IPC_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

39726 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in
 39727 the *sbuf* array. The number of processes that can execute concurrently without queuing is
 39728 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in
 39729 the program.

39730 The final call to *semop()* acquires the semaphore and waits until it is free; the SEM_UNDO
 39731 option releases the semaphore when the process exits, waiting until there are less than two
 39732 processes running concurrently.

```
39733 #include <sys/types.h>
39734 #include <stdio.h>
39735 #include <sys/ipc.h>
39736 #include <sys/sem.h>
39737 #include <sys/stat.h>
39738 #include <errno.h>
39739 #include <unistd.h>
39740 #include <stdlib.h>
```



```

39741     #include <pwd.h>
39742     #include <fcntl.h>
39743     #include <limits.h>
39744     ...
39745     key_t semkey;
39746     int semid, pfd, fv;
39747     struct sembuf sbuf;
39748     char *lgn;
39749     char filename[PATH_MAX+1];
39750     struct stat outstat;
39751     struct passwd *pw;
39752     ...
39753     /* Get unique key for semaphore. */
39754     if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39755         perror("IPC error: ftok"); exit(1);
39756     }
39757     /* Get semaphore ID associated with this key. */
39758     if ((semid = semget(semkey, 0, 0)) == -1) {
39759         /* Semaphore does not exist - Create. */
39760         if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39761             S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39762         {
39763             /* Initialize the semaphore. */
39764             sbuf.sem_num = 0;
39765             sbuf.sem_op = 2; /* This is the number of runs without queuing. */
39766             sbuf.sem_flg = 0;
39767             if (semop(semid, &sbuf, 1) == -1) {
39768                 perror("IPC error: semop"); exit(1);
39769             }
39770         }
39771         else if (errno == EEXIST) {
39772             if ((semid = semget(semkey, 0, 0)) == -1) {
39773                 perror("IPC error 1: semget"); exit(1);
39774             }
39775         }
39776         else {
39777             perror("IPC error 2: semget"); exit(1);
39778         }
39779     }
39780     ...
39781     sbuf.sem_num = 0;
39782     sbuf.sem_op = -1;
39783     sbuf.sem_flg = SEM_UNDO;
39784     if (semop(semid, &sbuf, 1) == -1) {
39785         perror("IPC Error: semop"); exit(1);
39786     }

```

39787 APPLICATION USAGE

39788 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
39789 Application developers who need to use IPC should design their applications so that modules
39790 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the
39791 alternative interfaces.

39792 **RATIONALE**

39793 None.

39794 **FUTURE DIRECTIONS**

39795 None.

39796 **SEE ALSO**

39797 Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit()*, *fork()*, *semctl()*, *semget()*,
39798 *sem_close()*, *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_unlink()*,
39799 *sem_wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/ipc.h>**, **<sys/sem.h>**,
39800 **<sys/types.h>**

39801 **CHANGE HISTORY**

39802 First released in Issue 2. Derived from Issue 2 of the SVID.

39803 **Issue 5**

39804 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
39805 DIRECTIONS to a new APPLICATION USAGE section.

39806 **NAME**

39807 send — send a message on a socket

39808 **SYNOPSIS**

39809 #include <sys/socket.h>

39810 ssize_t send(int *socket*, const void **buffer*, size_t *length*, int *flags*);39811 **DESCRIPTION**39812 The *send()* function shall initiate transmission of a message from the specified socket to its peer.39813 The *send()* function shall send a message only when the socket is connected (including when the
39814 peer of a connectionless socket has been set via *connect()*).39815 The *send()* function takes the following arguments:39816 *socket* Specifies the socket file descriptor.39817 *buffer* Points to the buffer containing the message to send.39818 *length* Specifies the length of the message in bytes.39819 *flags* Specifies the type of message transmission. Values of this argument are
39820 formed by logically OR'ing zero or more of the following flags:

39821 MSG_EOR Terminates a record (if supported by the protocol).

39822 MSG_OOB Sends out-of-band data on sockets that support out-of-band
39823 communications. The significance and semantics of out-of-
39824 band data are protocol-specific.39825 The length of the message to be sent is specified by the *length* argument. If the message is too
39826 long to pass through the underlying protocol, *send()* shall fail and no data shall be transmitted.39827 Successful completion of a call to *send()* does not guarantee delivery of the message. A return
39828 value of -1 indicates only locally-detected errors.39829 If space is not available at the sending socket to hold the message to be transmitted, and the
39830 socket file descriptor does not have O_NONBLOCK set, *send()* shall block until space is
39831 available. If space is not available at the sending socket to hold the message to be transmitted,
39832 and the socket file descriptor does have O_NONBLOCK set, *send()* shall fail. The *select()* and
39833 *poll()* functions can be used to determine when it is possible to send more data.39834 The socket in use may require the process to have appropriate privileges to use the *send()*
39835 function.39836 **RETURN VALUE**39837 Upon successful completion, *send()* shall return the number of bytes sent. Otherwise, -1 shall be
39838 returned and *errno* set to indicate the error.39839 **ERRORS**39840 The *send()* function shall fail if:

39841 [EAGAIN] or [EWOULDBLOCK]

39842 The socket's file descriptor is marked O_NONBLOCK and the requested
39843 operation would block.39844 [EBADF] The *socket* argument is not a valid file descriptor.

39845 [ECONNRESET] A connection was forcibly closed by a peer.

39846 [EDESTADDRREQ]

39847 The socket is not connection-mode and no peer address is set.

39848	[EINTR]	A signal interrupted <i>send()</i> before any data was transmitted.
39849	[EMSGSIZE]	The message is too large to be sent all at once, as the socket requires.
39850	[ENOTCONN]	The socket is not connected or otherwise has not had the peer pre-specified.
39851	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
39852	[EOPNOTSUPP]	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
39854	[EPIPE]	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type <code>SOCK_STREAM</code> , the <code>SIGPIPE</code> signal is generated to the calling thread.
39855		
39856		
39857		The <i>send()</i> function may fail if:
39858	[EACCES]	The calling process does not have the appropriate privileges.
39859	[EIO]	An I/O error occurred while reading from or writing to the file system.
39860	[ENETDOWN]	The local network interface used to reach the destination is down.
39861	[ENETUNREACH]	
39862		No route to the network is present.
39863	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.

39864 EXAMPLES

39865 None.

39866 APPLICATION USAGE

39867 The *send()* function is equivalent to *sendto()* with a null pointer *dest_len* argument, and to *write()*
 39868 if no flags are used.

39869 RATIONALE

39870 None.

39871 FUTURE DIRECTIONS

39872 None.

39873 SEE ALSO

39874 *connect()*, *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *sendmsg()*, *sendto()*,
 39875 *setsockopt()*, *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 39876 **<sys/socket.h>**

39877 CHANGE HISTORY

39878 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39879 **NAME**

39880 sendmsg — send a message on a socket using a message structure

39881 **SYNOPSIS**

39882 #include <sys/socket.h>

39883 ssize_t sendmsg(int *socket*, const struct msghdr **message*, int *flags*);39884 **DESCRIPTION**

39885 The *sendmsg()* function shall send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message shall be sent to the address specified by **msghdr**. If the socket is connection-mode, the destination address in **msghdr** shall be ignored.

39889 The *sendmsg()* function takes the following arguments:

39890	<i>socket</i>	Specifies the socket file descriptor.
39891	<i>message</i>	Points to a msghdr structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored.
39892		
39893		
39894	<i>flags</i>	Specifies the type of message transmission. The application may specify 0 or the following flag:
39895		
39896	MSG_EOR	Terminates a record (if supported by the protocol).
39897	MSG_OOB	Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
39898		
39899		

39900 The *msg_iov* and *msg_iovlen* fields of *message* specify zero or more buffers containing the data to be sent. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated by *msg_iov* is sent in turn.

39905 Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A return value of -1 indicates only locally-detected errors.

39907 If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have **O_NONBLOCK** set, the *sendmsg()* function shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have **O_NONBLOCK** set, the *sendmsg()* function shall fail.

39912 If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendmsg()* shall fail if the **SO_BROADCAST** option is not set for the socket.

39914 The socket in use may require the process to have appropriate privileges to use the *sendmsg()* function.

39916 **RETURN VALUE**

39917 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise, -1 shall be returned and *errno* set to indicate the error.

39919 **ERRORS**39920 The *sendmsg()* function shall fail if:

39921 [EAGAIN] or [EWOULDBLOCK]

39922 The socket's file descriptor is marked **O_NONBLOCK** and the requested

39923		operation would block.
39924	[EAFNOSUPPORT]	
39925		Addresses in the specified address family cannot be used with this socket.
39926	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
39927	[ECONNRESET]	A connection was forcibly closed by a peer.
39928	[EINTR]	A signal interrupted <i>sendmsg()</i> before any data was transmitted.
39929	[EINVAL]	The sum of the <i>iov_len</i> values overflows an ssize_t .
39930	[EMSGSIZE]	The message is too large to be sent all at once (as the socket requires), or the <i>msg_iovlen</i> member of the msghdr structure pointed to by <i>message</i> is less than or equal to 0 or is greater than {IOV_MAX}.
39931		
39932		
39933	[ENOTCONN]	The socket is connection-mode but is not connected.
39934	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
39935	[EOPNOTSUPP]	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
39936		
39937	[EPIPE]	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.
39938		
39939		
39940		If the address family of the socket is AF_UNIX, then <i>sendmsg()</i> shall fail if:
39941	[EIO]	An I/O error occurred while reading from or writing to the file system.
39942	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname in the socket address.
39943		
39944	[ENAMETOOLONG]	
39945		A component of a pathname exceeded {NAME_MAX} characters, or an entire pathname exceeded {PATH_MAX} characters.
39946		
39947	[ENOENT]	A component of the pathname does not name an existing file or the path name is an empty string.
39948		
39949	[ENOTDIR]	A component of the path prefix of the pathname in the socket address is not a directory.
39950		
39951		The <i>sendmsg()</i> function may fail if:
39952	[EACCES]	Search permission is denied for a component of the path prefix; or write access to the named socket is denied.
39953		
39954	[EDESTADDRREQ]	
39955		The socket is not connection-mode and does not have its peer address set, and no destination address was specified.
39956		
39957	[EHOSTUNREACH]	
39958		The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).
39959		
39960	[EIO]	An I/O error occurred while reading from or writing to the file system.
39961	[EISCONN]	A destination address was specified and the socket is already connected.
39962	[ENETDOWN]	The local network interface used to reach the destination is down.

39963 [ENETUNREACH]
39964 No route to the network is present.

39965 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

39966 [ENOMEM] Insufficient memory was available to fulfill the request.

39967 If the address family of the socket is AF_UNIX, then *sendmsg()* may fail if:

39968 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
39969 resolution of the pathname in the socket address.

39970 [ENAMETOOLONG]
39971 Pathname resolution of a symbolic link produced an intermediate result
39972 whose length exceeds {PATH_MAX}.

39973 **EXAMPLES**
39974 Done.

39975 **APPLICATION USAGE**
39976 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.

39977 **RATIONALE**
39978 None.

39979 **FUTURE DIRECTIONS**
39980 None.

39981 **SEE ALSO**
39982 *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendto()*, *setsockopt()*,
39983 *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>

39984 **CHANGE HISTORY**
39985 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39986 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
39987 [ELOOP] error condition is added.

39988 **NAME**

39989 sendto — send a message on a socket

39990 **SYNOPSIS**

39991 #include <sys/socket.h>

```
39992        ssize_t sendto(int socket, const void *message, size_t length,
39993                        int flags, const struct sockaddr *dest_addr,
39994                        socklen_t dest_len);
```

39995 **DESCRIPTION**

39996 The *sendto()* function shall send a message through a connection-mode or connectionless-mode
 39997 socket. If the socket is connectionless-mode, the message shall be sent to the address specified by
 39998 *dest_addr*. If the socket is connection-mode, *dest_addr* shall be ignored.

39999 The *sendto()* function takes the following arguments:

40000	<i>socket</i>	Specifies the socket file descriptor.
40001	<i>message</i>	Points to a buffer containing the message to be sent.
40002	<i>length</i>	Specifies the size of the message in bytes.
40003	<i>flags</i>	Specifies the type of message transmission. Values of this argument are
40004		formed by logically OR'ing zero or more of the following flags:
40005	MSG_EOR	Terminates a record (if supported by the protocol).
40006	MSG_OOB	Sends out-of-band data on sockets that support out-of-band
40007		data. The significance and semantics of out-of-band data are
40008		protocol-specific.
40009	<i>dest_addr</i>	Points to a sockaddr structure containing the destination address. The length
40010		and format of the address depend on the address family of the socket.
40011	<i>dest_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>dest_addr</i>
40012		argument.

40013 If the socket protocol supports broadcast and the specified address is a broadcast address for the
 40014 socket protocol, *sendto()* shall fail if the SO_BROADCAST option is not set for the socket.

40015 The *dest_addr* argument specifies the address of the target. The *length* argument specifies the
 40016 length of the message.

40017 Successful completion of a call to *sendto()* does not guarantee delivery of the message. A return
 40018 value of -1 indicates only locally-detected errors.

40019 If space is not available at the sending socket to hold the message to be transmitted and the
 40020 socket file descriptor does not have O_NONBLOCK set, *sendto()* shall block until space is
 40021 available. If space is not available at the sending socket to hold the message to be transmitted
 40022 and the socket file descriptor does have O_NONBLOCK set, *sendto()* shall fail.

40023 The socket in use may require the process to have appropriate privileges to use the *sendto()*
 40024 function.

40025 **RETURN VALUE**

40026 Upon successful completion, *sendto()* shall return the number of bytes sent. Otherwise, -1 shall
 40027 be returned and *errno* set to indicate the error.

40028 **ERRORS**

40029	The <i>sendto()</i> function shall fail if:	
40030	[EAFNOSUPPORT]	Addresses in the specified address family cannot be used with this socket.
40031		
40032	[EAGAIN] or [EWOULDBLOCK]	The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.
40033		
40034		
40035	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
40036	[ECONNRESET]	A connection was forcibly closed by a peer.
40037	[EINTR]	A signal interrupted <i>sendto()</i> before any data was transmitted.
40038	[EMSGSIZE]	The message is too large to be sent all at once, as the socket requires.
40039	[ENOTCONN]	The socket is connection-mode but is not connected.
40040	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
40041	[EOPNOTSUPP]	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
40042		
40043	[EPIPE]	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.
40044		
40045		
40046	If the address family of the socket is AF_UNIX, then <i>sendto()</i> shall fail if:	
40047	[EIO]	An I/O error occurred while reading from or writing to the file system.
40048	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname in the socket address.
40049		
40050	[ENAMETOOLONG]	A component of a pathname exceeded {NAME_MAX} characters, or an entire pathname exceeded {PATH_MAX} characters.
40051		
40052		
40053	[ENOENT]	A component of the pathname does not name an existing file or the pathname is an empty string.
40054		
40055	[ENOTDIR]	A component of the path prefix of the pathname in the socket address is not a directory.
40056		
40057	The <i>sendto()</i> function may fail if:	
40058	[EACCES]	Search permission is denied for a component of the path prefix; or write access to the named socket is denied.
40059		
40060	[EDESTADDRREQ]	The socket is not connection-mode and does not have its peer address set, and no destination address was specified.
40061		
40062		
40063	[EHOSTUNREACH]	The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).
40064		
40065		
40066	[EINVAL]	The <i>dest_len</i> argument is not a valid length for the address family.
40067	[EIO]	An I/O error occurred while reading from or writing to the file system.

40068	[EISCONN]	A destination address was specified and the socket is already connected. This error may or may not be returned for connection mode sockets.
40069		
40070	[ENETDOWN]	The local network interface used to reach the destination is down.
40071	[ENETUNREACH]	
40072		No route to the network is present.
40073	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
40074	[ENOMEM]	Insufficient memory was available to fulfill the request.
40075		If the address family of the socket is AF_UNIX, then <i>sendto()</i> may fail if:
40076	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in the socket address.
40077		
40078	[ENAMETOOLONG]	
40079		Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
40080		
40081	EXAMPLES	
40082		None.
40083	APPLICATION USAGE	
40084		The <i>select()</i> and <i>poll()</i> functions can be used to determine when it is possible to send more data.
40085	RATIONALE	
40086		None.
40087	FUTURE DIRECTIONS	
40088		None.
40089	SEE ALSO	
40090		<i>getsockopt()</i> , <i>poll()</i> , <i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>select()</i> , <i>send()</i> , <i>sendmsg()</i> , <i>setsockopt()</i> ,
40091		<i>shutdown()</i> , <i>socket()</i> , the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>
40092	CHANGE HISTORY	
40093		First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
40094		The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.
40095		

40096 **NAME**

40097 setbuf — assign buffering to a stream

40098 **SYNOPSIS**

40099 #include <stdio.h>

40100 void setbuf(FILE *restrict stream, char *restrict buf);

40101 **DESCRIPTION**

40102 cx The functionality described on this reference page is aligned with the ISO C standard. Any
40103 conflict between the requirements described here and the ISO C standard is unintentional. This
40104 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

40105 Except that it returns no value, the function call:

40106 setbuf(stream, buf)

40107 shall be equivalent to:

40108 setvbuf(stream, buf, _IOFBF, BUFSIZ)

40109 if *buf* is not a null pointer, or to:

40110 setvbuf(stream, buf, _IONBF, BUFSIZ)

40111 if *buf* is a null pointer.40112 **RETURN VALUE**40113 The *setbuf()* function shall not return a value.40114 **ERRORS**

40115 No errors are defined.

40116 **EXAMPLES**

40117 None.

40118 **APPLICATION USAGE**

40119 A common source of error is allocating buffer space as an “automatic” variable in a code block,
40120 and then failing to close the stream in the same block.

40121 With *setbuf()*, allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ
40122 bytes are used for the buffer area.

40123 **RATIONALE**

40124 None.

40125 **FUTURE DIRECTIONS**

40126 None.

40127 **SEE ALSO**40128 *fopen()*, *setvbuf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>40129 **CHANGE HISTORY**

40130 First released in Issue 1. Derived from Issue 1 of the SVID.

40131 **Issue 6**40132 The prototype for *setbuf()* is updated for alignment with the ISO / IEC 9899: 1999 standard.

40133 **NAME**

40134 setcontext — set current user context

40135 **SYNOPSIS**

40136 xSI #include <ucontext.h>

40137 int setcontext(const ucontext_t *ucp);

40138

40139 **DESCRIPTION**

40140 Refer to *getcontext()*.

40141 **NAME**

40142 setegid — set the effective group ID

40143 **SYNOPSIS**

40144 #include <unistd.h>

40145 int setegid(gid_t *gid*);40146 **DESCRIPTION**

40147 If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate
40148 privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group
40149 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

40150 The *setegid()* function shall not affect the supplementary group list in any way.

40151 **RETURN VALUE**

40152 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
40153 indicate the error.

40154 **ERRORS**

40155 The *setegid()* function shall fail if:

40156 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
40157 implementation.

40158 [EPERM] The process does not have appropriate privileges and *gid* does not match the
40159 real group ID or the saved set-group-ID.

40160 **EXAMPLES**

40161 None.

40162 **APPLICATION USAGE**

40163 None.

40164 **RATIONALE**

40165 Refer to the RATIONALE section in *setuid()*.

40166 **FUTURE DIRECTIONS**

40167 None.

40168 **SEE ALSO**

40169 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the
40170 Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

40171 **CHANGE HISTORY**

40172 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40173 **NAME**

40174 setenv — add or change environment variable

40175 **SYNOPSIS**

40176 CX #include <stdlib.h>

40177 int setenv(const char *envname, const char *envval, int overwrite);

40178

40179 **DESCRIPTION**

40180 The *setenv()* function shall update or add a variable in the environment of the calling process.
 40181 The *envname* argument points to a string containing the name of an environment variable to be
 40182 added or altered. The environment variable shall be set to the value to which *envval* points. The
 40183 function shall fail if *envname* points to a string which contains an '=' character. If the
 40184 environment variable named by *envname* already exists and the value of *overwrite* is non-zero,
 40185 the function shall return success and the environment shall be updated. If the environment
 40186 variable named by *envname* already exists and the value of *overwrite* is zero, the function shall
 40187 return success and the environment shall remain unchanged.

40188 If the application modifies *environ* or the pointers to which it points, the behavior of *setenv()* is
 40189 undefined. The *setenv()* function shall update the list of pointers to which *environ* points.

40190 The strings described by *envname* and *envval* are copied by this function.

40191 The *setenv()* function need not be reentrant. A function that is not required to be reentrant is not
 40192 required to be thread-safe.

40193 **RETURN VALUE**

40194 Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to
 40195 indicate the error, and the environment shall be unchanged.

40196 **ERRORS**40197 The *setenv()* function shall fail if:

40198 [EINVAL] The *name* argument is a null pointer, points to an empty string, or points to a
 40199 string containing an '=' character.

40200 [ENOMEM] Insufficient memory was available to add a variable or its value to the
 40201 environment.

40202 **EXAMPLES**

40203 None.

40204 **APPLICATION USAGE**40205 See *exec*, for restrictions on changing the environment in multi-threaded applications.40206 **RATIONALE**

40207 Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular,
 40208 if the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an
 40209 obsolete copy of the environment (as may any other copy of *environ*). However, other than the
 40210 aforementioned restriction, the developers of IEEE Std 1003.1-2001 intended that the traditional
 40211 method of walking through the environment by way of the *environ* pointer must be supported.

40212 It was decided that *setenv()* should be required by this revision because it addresses a piece of
 40213 missing functionality, and does not impose a significant burden on the implementor.

40214 There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()*
 40215 function should be required as a mandatory function. The *setenv()* function was chosen because
 40216 it permitted the implementation of the *unsetenv()* function to delete environmental variables,
 40217 without specifying an additional interface. The *putenv()* function is available as an XSI

40218 extension.

40219 The standard developers considered requiring that *setenv()* indicate an error when a call to it
40220 would result in exceeding {ARG_MAX}. The requirement was rejected since the condition might
40221 be temporary, with the application eventually reducing the environment size. The ultimate
40222 success or failure depends on the size at the time of a call to *exec*, which returns an indication of
40223 this error condition.

40224 **FUTURE DIRECTIONS**

40225 None.

40226 **SEE ALSO**

40227 *exec*, *getenv()*, *unsetenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>, |
40228 <sys/types.h>, <unistd.h>

40229 **CHANGE HISTORY**

40230 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40231 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH / TC1 / D6 / 55 is applied, adding references to *exec* in |
40232 the APPLICATION USAGE and SEE ALSO sections. |

40233 **NAME**

40234 seteuid — set effective user ID

40235 **SYNOPSIS**

40236 #include <unistd.h>

40237 int seteuid(uid_t uid);

40238 **DESCRIPTION**

40239 If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate
40240 privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID
40241 and saved set-user-ID shall remain unchanged.

40242 The *seteuid()* function shall not affect the supplementary group list in any way.

40243 **RETURN VALUE**

40244 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
40245 indicate the error.

40246 **ERRORS**

40247 The *seteuid()* function shall fail if:

40248 [EINVAL] The value of the *uid* argument is invalid and is not supported by the
40249 implementation.

40250 [EPERM] The process does not have appropriate privileges and *uid* does not match the
40251 real group ID or the saved set-group-ID.

40252 **EXAMPLES**

40253 None.

40254 **APPLICATION USAGE**

40255 None.

40256 **RATIONALE**

40257 Refer to the RATIONALE section in *setuid()*.

40258 **FUTURE DIRECTIONS**

40259 None.

40260 **SEE ALSO**

40261 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the
40262 Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

40263 **CHANGE HISTORY**

40264 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40265 **NAME**

40266 setgid — set-group-ID

40267 **SYNOPSIS**

40268 #include <unistd.h>

40269 int setgid(gid_t *gid*);40270 **DESCRIPTION**

40271 If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID, and the saved set-group-ID of the calling process to *gid*.

40273 If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved set-group-ID shall remain unchanged.

40276 The *setgid()* function shall not affect the supplementary group list in any way.

40277 Any supplementary group IDs of the calling process shall remain unchanged.

40278 **RETURN VALUE**

40279 Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

40281 **ERRORS**

40282 The *setgid()* function shall fail if:

40283 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
40284 implementation.

40285 [EPERM] The process does not have appropriate privileges and *gid* does not match the
40286 real group ID or the saved set-group-ID.

40287 **EXAMPLES**

40288 None.

40289 **APPLICATION USAGE**

40290 None.

40291 **RATIONALE**

40292 Refer to the RATIONALE section in *setuid()*.

40293 **FUTURE DIRECTIONS**

40294 None.

40295 **SEE ALSO**

40296 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*, *setreuid()*, *setuid()*, the
40297 Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

40298 **CHANGE HISTORY**

40299 First released in Issue 1. Derived from Issue 1 of the SVID.

40300 **Issue 6**

40301 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

40302 The following new requirements on POSIX implementations derive from alignment with the
40303 Single UNIX Specification:

- 40304 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
40305 required for conforming implementations of previous POSIX specifications, it was not
40306 required for UNIX applications.

- 40307 • Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS
- 40308 requirement.
- 40309 The following changes were made to align with the IEEE P1003.1a draft standard:
- 40310 • The effects of `setgid()` in processes without appropriate privileges are changed.
- 40311 • A requirement that the supplementary group list is not affected is added.

40312 **NAME**

40313 setgrent — reset the group database to the first entry

40314 **SYNOPSIS**

40315 xSI #include <grp.h>

40316 void setgrent(void);

40317

40318 **DESCRIPTION**40319 Refer to *endgrent()*.

40320 NAME

40321 sethostent — network host database functions

40322 SYNOPSIS

40323 #include <netdb.h>

40324 void sethostent(int *stayopen*);

40325 DESCRIPTION

40326 Refer to *endhostent()*.

40327 **NAME**

40328 setitimer — set the value of an interval timer

40329 **SYNOPSIS**

40330 xSI #include <sys/time.h>

40331 int setitimer(int *which*, const struct itimerval *restrict *value*,
40332 struct itimerval *restrict *ovalue*);

40333

40334 **DESCRIPTION**40335 Refer to *getitimer()*.

40336 **NAME**

40337 setjmp — set jump point for a non-local goto

40338 **SYNOPSIS**

40339 #include <setjmp.h>

40340 int setjmp(jmp_buf env);

40341 **DESCRIPTION**

40342 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 40343 conflict between the requirements described here and the ISO C standard is unintentional. This
 40344 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

40345 A call to *setjmp()* shall save the calling environment in its *env* argument for later use by
 40346 *longjmp()*.

40347 It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in
 40348 order to access an actual function, or a program defines an external identifier with the name
 40349 *setjmp*, the behavior is undefined.

40350 An application shall ensure that an invocation of *setjmp()* appears in one of the following
 40351 contexts only:

- 40352 • The entire controlling expression of a selection or iteration statement
- 40353 • One operand of a relational or equality operator with the other operand an integral constant
 40354 expression, with the resulting expression being the entire controlling expression of a
 40355 selection or iteration statement
- 40356 • The operand of a unary '!' operator with the resulting expression being the entire
 40357 controlling expression of a selection or iteration
- 40358 • The entire expression of an expression statement (possibly cast to **void**)

40359 If the invocation appears in any other context, the behavior is undefined.

40360 **RETURN VALUE**

40361 If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to
 40362 *longjmp()*, *setjmp()* shall return a non-zero value.

40363 **ERRORS**

40364 No errors are defined.

40365 **EXAMPLES**

40366 None.

40367 **APPLICATION USAGE**

40368 In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-
 40369 level subroutine of a program.

40370 **RATIONALE**

40371 None.

40372 **FUTURE DIRECTIONS**

40373 None.

40374 **SEE ALSO**

40375 *longjmp()*, *sigsetjmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**setjmp.h**>

40376 **CHANGE HISTORY**

40377 First released in Issue 1. Derived from Issue 1 of the SVID.

40378 **Issue 6**

40379 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

40380 **NAME**40381 setkey — set encoding key (**CRYPT**)40382 **SYNOPSIS**

40383 xSI #include <stdlib.h>

40384 void setkey(const char *key);

40385

40386 **DESCRIPTION**

40387 The *setkey()* function provides access to an implementation-defined encoding algorithm. The
40388 argument of *setkey()* is an array of length 64 bytes containing only the bytes with numerical
40389 value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is
40390 ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used
40391 with the algorithm to encode a string *block* passed to *encrypt()*.

40392 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to
40393 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on
40394 return, an error has occurred.

40395 The *setkey()* function need not be reentrant. A function that is not required to be reentrant is not
40396 required to be thread-safe.

40397 **RETURN VALUE**

40398 No values are returned.

40399 **ERRORS**40400 The *setkey()* function shall fail if:

40401 [ENOSYS] The functionality is not supported on this implementation.

40402 **EXAMPLES**

40403 None.

40404 **APPLICATION USAGE**

40405 Decoding need not be implemented in all environments. This is related to government
40406 restrictions in some countries on encryption and decryption routines. Historical practice has
40407 been to ship a different version of the encryption library without the decryption feature in the
40408 routines supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

40409 **RATIONALE**

40410 None.

40411 **FUTURE DIRECTIONS**

40412 None.

40413 **SEE ALSO**40414 *crypt()*, *encrypt()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>40415 **CHANGE HISTORY**

40416 First released in Issue 1. Derived from Issue 1 of the SVID.

40417 **Issue 5**40418 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

40419 **NAME**

40420 setlocale — set program locale

40421 **SYNOPSIS**

40422 #include <locale.h>

40423 char *setlocale(int *category*, const char **locale*);40424 **DESCRIPTION**

40425 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40426 conflict between the requirements described here and the ISO C standard is unintentional. This
 40427 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

40428 The *setlocale()* function selects the appropriate piece of the program's locale, as specified by the
 40429 *category* and *locale* arguments, and may be used to change or query the program's entire locale or
 40430 portions thereof. The value *LC_ALL* for *category* names the program's entire locale; other values
 40431 for *category* name only a part of the program's locale:

40432 *LC_COLLATE* Affects the behavior of regular expressions and the collation functions.

40433 *LC_CTYPE* Affects the behavior of regular expressions, character classification, character
 40434 conversion functions, and wide-character functions.

40435 CX *LC_MESSAGES* Affects what strings are expected by commands and utilities as affirmative or
 40436 negative responses.

40437 XSI It also affects what strings are given by commands and utilities as affirmative
 40438 or negative responses, and the content of messages.

40439 *LC_MONETARY* Affects the behavior of functions that handle monetary values.

40440 *LC_NUMERIC* Affects the behavior of functions that handle numeric values.

40441 *LC_TIME* Affects the behavior of the time conversion functions.

40442 The *locale* argument is a pointer to a character string containing the required setting of *category*.
 40443 The contents of this string are implementation-defined. In addition, the following preset values
 40444 of *locale* are defined for all settings of *category*:

40445 CX "POSIX" Specifies the minimal environment for C-language translation called the
 40446 POSIX locale. If *setlocale()* is not invoked, the POSIX locale is the default at
 40447 entry to *main()*.

40448 "C" Equivalent to "POSIX".

40449 CX "" Specifies an implementation-defined native environment. This corresponds to
 40450 the value of the associated environment variables, *LC_** and *LANG*; see the
 40451 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale and the
 40452 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment
 40453 Variables.

40454 A null pointer Used to direct *setlocale()* to query the current internationalized environment
 40455 and return the name of the locale.

40456 THR The locale state is common to all threads within a process.

40457 **RETURN VALUE**

40458 Upon successful completion, *setlocale()* shall return the string associated with the specified
 40459 category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the program's
 40460 locale is not changed.

40461 A null pointer for *locale* causes *setlocale()* to return a pointer to the string associated with the
 40462 *category* for the program's current locale. The program's locale shall not be changed.

40463 The string returned by *setlocale()* is such that a subsequent call with that string and its associated
 40464 *category* shall restore that part of the program's locale. The application shall not modify the string
 40465 returned which may be overwritten by a subsequent call to *setlocale()*.

40466 ERRORS

40467 No errors are defined.

40468 EXAMPLES

40469 None.

40470 APPLICATION USAGE

40471 The following code illustrates how a program can initialize the international environment for
 40472 one language, while selectively modifying the program's locale such that regular expressions
 40473 and string operations can be applied to text recorded in a different language:

```
40474 setlocale(LC_ALL, "De");
40475 setlocale(LC_COLLATE, "Fr@dict");
```

40476 Internationalized programs must call *setlocale()* to initiate a specific language operation. This can
 40477 be done by calling *setlocale()* as follows:

```
40478 setlocale(LC_ALL, "");
```

40479 Changing the setting of *LC_MESSAGES* has no effect on catalogs that have already been opened
 40480 by calls to *catopen()*.

40481 RATIONALE

40482 The ISO C standard defines a collection of functions to support internationalization. One of the
 40483 most significant aspects of these functions is a facility to set and query the *international*
 40484 *environment*. The international environment is a repository of information that affects the
 40485 behavior of certain functionality, namely:

- 40486 1. Character handling
- 40487 2. Collating
- 40488 3. Date/time formatting
- 40489 4. Numeric editing
- 40490 5. Monetary formatting
- 40491 6. Messaging

40492 The *setlocale()* function provides the application developer with the ability to set all or portions,
 40493 called *categories*, of the international environment. These categories correspond to the areas of
 40494 functionality mentioned above. The syntax for *setlocale()* is as follows:

```
40495 char *setlocale(int category, const char *locale);
```

40496 where *category* is the name of one of following categories, namely:

```
40497 LC_COLLATE
40498 LC_CTYPE
40499 LC_MESSAGES
40500 LC_MONETARY
40501 LC_NUMERIC
40502 LC_TIME
```

In addition, a special value called *LC_ALL* directs *setlocale()* to set all categories.

There are two primary uses of *setlocale()*:

1. Querying the international environment to find out what it is set to
2. Setting the international environment, or *locale*, to a specific value

The behavior of *setlocale()* in these two areas is described below. Since it is difficult to describe the behavior in words, examples are used to illustrate the behavior of specific uses.

To query the international environment, *setlocale()* is invoked with a specific category and the NULL pointer as the locale. The NULL pointer is a special directive to *setlocale()* that tells it to query rather than set the international environment. The following syntax is used to query the name of the international environment:

```
setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
          LC_NUMERIC, LC_TIME}, (char *) NULL);
```

The *setlocale()* function shall return the string corresponding to the current international environment. This value may be used by a subsequent call to *setlocale()* to reset the international environment to this value. However, it should be noted that the return value from *setlocale()* may be a pointer to a static area within the function and is not guaranteed to remain unchanged (that is, it may be modified by a subsequent call to *setlocale()*). Therefore, if the purpose of calling *setlocale()* is to save the value of the current international environment so it can be changed and reset later, the return value should be copied to an array of **char** in the calling program.

There are three ways to set the international environment with *setlocale()*:

setlocale(category, string)

This usage sets a specific *category* in the international environment to a specific value corresponding to the value of the *string*. A specific example is provided below:

```
setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

In this example, all categories of the international environment are set to the locale corresponding to the string "fr_FR.ISO-8859-1", or to the French language as spoken in France using the ISO/IEC 8859-1:1998 standard codeset.

If the string does not correspond to a valid locale, *setlocale()* shall return a NULL pointer and the international environment is not changed. Otherwise, *setlocale()* shall return the name of the locale just set.

setlocale(category, "C")

The ISO C standard states that one locale must exist on all conforming implementations. The name of the locale is C and corresponds to a minimal international environment needed to support the C programming language.

setlocale(category, "")

This sets a specific category to an implementation-defined default. This corresponds to the value of the environment variables.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *iswalnum()*, *iswalphalpha()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *isxdigit()*,

40547 *localeconv()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *nl_langinfo()*, *printf()*, *scanf()*, *setlocale()*, *strcoll()*,
40548 *strerror()*, *strfmon()*, *strtod()*, *strxfrm()*, *tolower()*, *toupper()*, *towlower()*, *towupper()*, *wscoll()*,
40549 *wctod()*, *wctombs()*, *wcsxfrm()*, *wctomb()*, the Base Definitions volume of IEEE Std 1003.1-2001,
40550 **<langinfo.h>**, **<locale.h>**

40551 **CHANGE HISTORY**

40552 First released in Issue 3.

40553 **Issue 5**

40554 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

40555 **Issue 6**

40556 Extensions beyond the ISO C standard are marked.

40557 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

40558 **NAME**

40559 setlogmask — set the log priority mask

40560 **SYNOPSIS**

40561 xSI #include <syslog.h>

40562 int setlogmask(int maskpri);

40563

40564 **DESCRIPTION**40565 Refer to *closelog()*.

40566 NAME

40567 setnetent — network database function

40568 SYNOPSIS

40569 #include <netdb.h>

40570 void setnetent(int stayopen);

40571 DESCRIPTION

40572 Refer to *endnetent()*.

40573 **NAME**

40574 setpgid — set process group ID for job control

40575 **SYNOPSIS**

40576 #include <unistd.h>

40577 int setpgid(pid_t pid, pid_t pgid);

40578 **DESCRIPTION**

40579 The *setpgid()* function shall either join an existing process group or create a new process group
 40580 within the session of the calling process. The process group ID of a session leader shall not
 40581 change. Upon successful completion, the process group ID of the process with a process ID that
 40582 matches *pid* shall be set to *pgid*. As a special case, if *pid* is 0, the process ID of the calling process
 40583 shall be used. Also, if *pgid* is 0, the process ID of the indicated process shall be used.

40584 **RETURN VALUE**

40585 Upon successful completion, *setpgid()* shall return 0; otherwise, -1 shall be returned and *errno*
 40586 shall be set to indicate the error.

40587 **ERRORS**40588 The *setpgid()* function shall fail if:

40589 [EACCES] The value of the *pid* argument matches the process ID of a child process of the
 40590 calling process and the child process has successfully executed one of the *exec*
 40591 functions.

40592 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by
 40593 the implementation.

40594 [EPERM] The process indicated by the *pid* argument is a session leader.

40595 [EPERM] The value of the *pid* argument matches the process ID of a child process of the
 40596 calling process and the child process is not in the same session as the calling
 40597 process.

40598 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of
 40599 the process indicated by the *pid* argument and there is no process with a
 40600 process group ID that matches the value of the *pgid* argument in the same
 40601 session as the calling process.

40602 [ESRCH] The value of the *pid* argument does not match the process ID of the calling
 40603 process or of a child process of the calling process.

40604 **EXAMPLES**

40605 None.

40606 **APPLICATION USAGE**

40607 None.

40608 **RATIONALE**

40609 The *setpgid()* function shall group processes together for the purpose of signaling, placement in
 40610 foreground or background, and other job control actions.

40611 The *setpgid()* function is similar to the *setpgrp()* function of 4.2 BSD, except that 4.2 BSD allowed
 40612 the specified new process group to assume any value. This presents certain security problems
 40613 and is more flexible than necessary to support job control.

40614 To provide tighter security, *setpgid()* only allows the calling process to join a process group
 40615 already in use inside its session or create a new process group whose process group ID was
 40616 equal to its process ID.

When a job control shell spawns a new job, the processes in the job must be placed into a new process group via *setpgid()*. There are two timing constraints involved in this action:

1. The new process must be placed in the new process group before the appropriate program is launched via one of the *exec* functions.
2. The new process must be placed in the new process group before the shell can correctly send signals to the new process group.

To address these constraints, the following actions are performed. The new processes call *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization property was considered, but it was decided instead to merely allow the parent shell process to adjust the process group of its child processes via *setpgid()*. Both timing constraints are now satisfied by having both the parent shell and the child attempt to adjust the process group of the child process; it does not matter which succeeds first.

Since it would be confusing to an application to have its process group change after it began executing (that is, after *exec*), and because the child process would already have adjusted its process group before this, the [EACCES] error was added to disallow this.

One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original process group (the one in effect when the job control shell was executed). A job control shell does this before returning control back to its parent when it is terminating or suspending itself as a way of restoring its job control “state” back to what its parent would expect. (Note that the original process group of the job control shell typically matches the process group of its parent, but this is not necessarily always the case.)

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getpgrp()*, *setsid()*, *tcsetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The *setpgid()* function is mandatory since `_POSIX_JOB_CONTROL` is required to be defined in this issue. This is a FIPS requirement.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in the DESCRIPTION from “the process group ID of the indicated process shall be used” to “the process ID of the indicated process shall be used”. This change reverts the wording to as in the ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

40661 **NAME**

40662 setpgrp — set the process group ID

40663 **SYNOPSIS**

40664 XSI #include <unistd.h>

40665 pid_t setpgrp(void);

40666

40667 **DESCRIPTION**

40668 If the calling process is not already a session leader, *setpgrp()* sets the process group ID of the
40669 calling process to the process ID of the calling process. If *setpgrp()* creates a new session, then
40670 the new session has no controlling terminal.

40671 The *setpgrp()* function has no effect when the calling process is a session leader.

40672 **RETURN VALUE**

40673 Upon completion, *setpgrp()* shall return the process group ID.

40674 **ERRORS**

40675 No errors are defined.

40676 **EXAMPLES**

40677 None.

40678 **APPLICATION USAGE**

40679 None.

40680 **RATIONALE**

40681 None.

40682 **FUTURE DIRECTIONS**

40683 None.

40684 **SEE ALSO**

40685 *exec*, *fork()*, *getpid()*, *getsid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of
40686 IEEE Std 1003.1-2001, <unistd.h>

40687 **CHANGE HISTORY**

40688 First released in Issue 4, Version 2.

40689 **Issue 5**

40690 Moved from X/OPEN UNIX extension to BASE.

40691 NAME

40692 setpriority — set the nice value

40693 SYNOPSIS

40694 xSI #include <sys/resource.h>

40695 int setpriority(int *which*, id_t *who*, int *nice*);

40696

40697 DESCRIPTION

40698 Refer to *getpriority()*.

40699 **NAME**

40700 setprotoent — network protocol database functions

40701 **SYNOPSIS**

40702 #include <netdb.h>

40703 void setprotoent(int *stayopen*);40704 **DESCRIPTION**40705 Refer to *endprotoent()*.

40706 NAME

40707 setpwent — user database function

40708 SYNOPSIS

40709 xSI #include <pwd.h>

40710 void setpwent(void);

40711

40712 DESCRIPTION

40713 Refer to *endpwent()*.

40714 **NAME**

40715 setregid — set real and effective group IDs

40716 **SYNOPSIS**

40717 xSI #include <unistd.h>

40718 int setregid(gid_t rgid, gid_t egid);

40719

40720 **DESCRIPTION**40721 The *setregid()* function shall set the real and effective group IDs of the calling process.40722 If *rgid* is *-1*, the real group ID shall not be changed; if *egid* is *-1*, the effective group ID shall not
40723 be changed.

40724 The real and effective group IDs may be set to different values in the same call.

40725 Only a process with appropriate privileges can set the real group ID and the effective group ID
40726 to any valid value.40727 A non-privileged process can set either the real group ID to the saved set-group-ID from one of
40728 the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group
40729 ID.

40730 Any supplementary group IDs of the calling process remain unchanged.

40731 **RETURN VALUE**40732 Upon successful completion, 0 shall be returned. Otherwise, *-1* shall be returned and *errno* set to
40733 indicate the error, and neither of the group IDs are changed.40734 **ERRORS**40735 The *setregid()* function shall fail if:40736 [EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.40737 [EPERM] The process does not have appropriate privileges and a change other than
40738 changing the real group ID to the saved set-group-ID, or changing the
40739 effective group ID to the real group ID or the saved set-group-ID, was
40740 requested.40741 **EXAMPLES**

40742 None.

40743 **APPLICATION USAGE**40744 If a set-group-ID process sets its effective group ID to its real group ID, it can still set its effective
40745 group ID back to the saved set-group-ID.40746 **RATIONALE**

40747 None.

40748 **FUTURE DIRECTIONS**

40749 None.

40750 **SEE ALSO**40751 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setreuid()*, *setuid()*, the
40752 Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>40753 **CHANGE HISTORY**

40754 First released in Issue 4, Version 2.

40755 **Issue 5**

40756 Moved from X/OPEN UNIX extension to BASE.

40757 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the
40758 *exec* family of functions, not just *execve()*.

40759 **NAME**

40760 setreuid — set real and effective user IDs

40761 **SYNOPSIS**

```
40762 xSI      #include <unistd.h>
40763          int setreuid(uid_t ruid, uid_t euid);
40764
```

40765 **DESCRIPTION**

40766 The *setreuid()* function shall set the real and effective user IDs of the current process to the
 40767 values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is -1 , the corresponding effective
 40768 or real user ID of the current process shall be left unchanged.

40769 A process with appropriate privileges can set either ID to any value. An unprivileged process
 40770 can only set the effective user ID if the *euid* argument is equal to either the real, effective, or
 40771 saved user ID of the process.

40772 It is unspecified whether a process without appropriate privileges is permitted to change the real
 40773 user ID to match the current real, effective, or saved set-user-ID of the process.

40774 **RETURN VALUE**

40775 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 40776 indicate the error.

40777 **ERRORS**

40778 The *setreuid()* function shall fail if:

40779	[EINVAL]	The value of the <i>ruid</i> or <i>euid</i> argument is invalid or out-of-range.
40780	[EPERM]	The current process does not have appropriate privileges, and either an attempt was made to change the effective user ID to a value other than the real user ID or the saved set-user-ID or an attempt was made to change the real user ID to a value not permitted by the implementation.

40784 **EXAMPLES**40785 **Setting the Effective User ID to the Real User ID**

40786 The following example sets the effective user ID of the calling process to the real user ID, so that
 40787 files created later will be owned by the current user.

```
40788 #include <unistd.h>
40789 #include <sys/types.h>
40790 ...
40791 setreuid(getuid(), getuid());
40792 ...
```

40793 **APPLICATION USAGE**

40794 None.

40795 **RATIONALE**

40796 None.

40797 **FUTURE DIRECTIONS**

40798 None.

40799 **SEE ALSO**

40800 *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setuid()*, the Base
40801 Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

40802 **CHANGE HISTORY**

40803 First released in Issue 4, Version 2.

40804 **Issue 5**

40805 Moved from X/OPEN UNIX extension to BASE.

40806 **NAME**

40807 setrlimit — control maximum resource consumption

40808 **SYNOPSIS**

40809 xSI #include <sys/resource.h>

40810 int setrlimit(int resource, const struct rlimit *rlp);

40811

40812 **DESCRIPTION**40813 Refer to *getrlimit()*.

40814 NAME

40815 setservent — network services database functions

40816 SYNOPSIS

40817 #include <netdb.h>

40818 void setservent(int *stayopen*);

40819 DESCRIPTION

40820 Refer to *endservent()*.

40821 NAME

40822 setsid — create session and set process group ID

40823 SYNOPSIS

40824 #include <unistd.h>

40825 pid_t setsid(void);

40826 DESCRIPTION

40827 The *setsid()* function shall create a new session, if the calling process is not a process group
40828 leader. Upon return the calling process shall be the session leader of this new session, shall be
40829 the process group leader of a new process group, and shall have no controlling terminal. The
40830 process group ID of the calling process shall be set equal to the process ID of the calling process.
40831 The calling process shall be the only process in the new process group and the only process in
40832 the new session.

40833 RETURN VALUE

40834 Upon successful completion, *setsid()* shall return the value of the new process group ID of the
40835 calling process. Otherwise, it shall return (**pid_t**)-1 and set *errno* to indicate the error.

40836 ERRORS

40837 The *setsid()* function shall fail if:

40838 [EPERM]	The calling process is already a process group leader, or the process group ID
40839	of a process other than the calling process matches the process ID of the
40840	calling process.

40841 EXAMPLES

40842 None.

40843 APPLICATION USAGE

40844 None.

40845 RATIONALE

40846 The *setsid()* function is similar to the *setpgp()* function of System V. System V, without job
40847 control, groups processes into process groups and creates new process groups via *setpgp()*; only
40848 one process group may be part of a login session.

40849 Job control allows multiple process groups within a login session. In order to limit job control
40850 actions so that they can only affect processes in the same login session, this volume of
40851 IEEE Std 1003.1-2001 adds the concept of a session that is created via *setsid()*. The *setsid()*
40852 function also creates the initial process group contained in the session. Additional process
40853 groups can be created via the *setpgid()* function. A System V process group would correspond to
40854 a POSIX System Interfaces session containing a single POSIX process group. Note that this
40855 function requires that the calling process not be a process group leader. The usual way to ensure
40856 this is true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function
40857 guarantees that the process ID of the new process does not match any existing process group ID.

40858 FUTURE DIRECTIONS

40859 None.

40860 SEE ALSO

40861 *getsid()*, *setpgid()*, *setpgp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>,
40862 <unistd.h>

40863 **CHANGE HISTORY**

40864 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

40865 **Issue 6**

40866 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

40867 The following new requirements on POSIX implementations derive from alignment with the
40868 Single UNIX Specification:

- 40869 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
40870 required for conforming implementations of previous POSIX specifications, it was not
40871 required for UNIX applications.

40872 **NAME**

40873 setsockopt — set the socket options

40874 **SYNOPSIS**

40875 #include <sys/socket.h>

```
40876       int setsockopt(int socket, int level, int option_name,
40877                      const void *option_value, socklen_t option_len);
```

40878 **DESCRIPTION**

40879 The *setsockopt()* function shall set the option specified by the *option_name* argument, at the
 40880 protocol level specified by the *level* argument, to the value pointed to by the *option_value*
 40881 argument for the socket associated with the file descriptor specified by the *socket* argument.

40882 The *level* argument specifies the protocol level at which the option resides. To set options at the
 40883 socket level, specify the *level* argument as SOL_SOCKET. To set options at other levels, supply
 40884 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate
 40885 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO_TCP
 40886 as defined in the <netinet/in.h> header.

40887 The *option_name* argument specifies a single option to set. The *option_name* argument and any
 40888 specified options are passed uninterpreted to the appropriate protocol module for
 40889 interpretations. The <sys/socket.h> header defines the socket-level options. The options are as
 40890 follows:

40891 SO_DEBUG Turns on recording of debugging information. This option enables or
 40892 disables debugging in the underlying protocol modules. This option takes
 40893 an **int** value. This is a Boolean option.

40894 SO_BROADCAST Permits sending of broadcast messages, if this is supported by the
 40895 protocol. This option takes an **int** value. This is a Boolean option.

40896 SO_REUSEADDR Specifies that the rules used in validating addresses supplied to *bind()*
 40897 should allow reuse of local addresses, if this is supported by the protocol.
 40898 This option takes an **int** value. This is a Boolean option.

40899 SO_KEEPALIVE Keeps connections active by enabling the periodic transmission of
 40900 messages, if this is supported by the protocol. This option takes an **int**
 40901 value.

40902 If the connected socket fails to respond to these messages, the connection
 40903 is broken and threads writing to that socket are notified with a SIGPIPE
 40904 signal. This is a Boolean option.

40905 SO_LINGER Lingers on a *close()* if data is present. This option controls the action
 40906 taken when unsent messages queue on a socket and *close()* is performed.
 40907 If SO_LINGER is set, the system shall block the process during *close()*
 40908 until it can transmit the data or until the time expires. If SO_LINGER is
 40909 not specified, and *close()* is issued, the system handles the call in a way
 40910 that allows the process to continue as quickly as possible. This option
 40911 takes a **linger** structure, as defined in the <sys/socket.h> header, to
 40912 specify the state of the option and linger interval.

40913 SO_OOBINLINE Leaves received out-of-band data (data marked urgent) inline. This
 40914 option takes an **int** value. This is a Boolean option.

40915 SO_SNDBUF Sets send buffer size. This option takes an **int** value.

40916	SO_RCVBUF	Sets receive buffer size. This option takes an int value.	
40917	SO_DONTROUTE	Requests that outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an int value. This is a Boolean option.	
40918			
40919			
40920			
40921			
40922	SO_RCVLOWAT	Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option takes an int value. Note that not all implementations allow this option to be set.	
40923			
40924			
40925			
40926			
40927			
40928			
40929			
40930	SO_RCVTIMEO	Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. It accepts a timeval structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is received. The default for this option is zero, which indicates that a receive operation shall not time out. This option takes a timeval structure. Note that not all implementations allow this option to be set.	
40931			
40932			
40933			
40934			
40935			
40936			
40937			
40938			
40939			
40940	SO_SNDLOWAT	Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an int value. Note that not all implementations allow this option to be set.	
40941			
40942			
40943			
40944			
40945	SO_SNDTIMEO	Sets the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is sent. The default for this option is zero, which indicates that a send operation shall not time out. This option stores a timeval structure. Note that not all implementations allow this option to be set.	
40946			
40947			
40948			
40949			
40950			
40951			
40952	For Boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.		
40953			
40954	Options at other protocol levels vary in format and name.		
40955	RETURN VALUE		
40956	Upon successful completion, <i>setsockopt()</i> shall return 0. Otherwise, -1 shall be returned and <i>errno</i> set to indicate the error.		
40957			
40958	ERRORS		
40959	The <i>setsockopt()</i> function shall fail if:		
40960	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.	
40961	[EDOM]	The send and receive timeout values are too big to fit into the timeout fields in the socket structure.	
40962			

40963	[EINVAL]	The specified option is invalid at the specified socket level or the socket has been shut down.
40964		
40965	[EISCONN]	The socket is already connected, and a specified option cannot be set while the socket is connected.
40966		
40967	[ENOPROTOOPT]	
40968		The option is not supported by the protocol.
40969	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
40970		The <i>setsockopt()</i> function may fail if:
40971	[ENOMEM]	There was insufficient memory available for the operation to complete.
40972	[ENOBUFS]	Insufficient resources are available in the system to complete the call.

EXAMPLES

40973
40974 None.

APPLICATION USAGE

40976 The *setsockopt()* function provides an application program with the means to control socket behavior. An application program can use *setsockopt()* to allocate buffer space, control timeouts, or permit socket data broadcasts. The **<sys/socket.h>** header defines the socket-level options available to *setsockopt()*.

40980 Options may exist at multiple protocol levels. The SO_ options are always present at the uppermost socket level.

RATIONALE

40982
40983 None.

FUTURE DIRECTIONS

40984
40985 None.

SEE ALSO

40987 Section 2.10 (on page 58), *bind()*, *endprotoent()*, *getsockopt()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<netinet/in.h>**, **<sys/socket.h>**

CHANGE HISTORY

40990 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

40991 NAME

40992 setstate — switch pseudo-random number generator state arrays

40993 SYNOPSIS

40994 xSI #include <stdlib.h>

40995 char *setstate(const char *state);

40996

40997 DESCRIPTION

40998 Refer to *initstate()*.

40999 **NAME**

41000 setuid — set user ID

41001 **SYNOPSIS**

41002 #include <unistd.h>

41003 int setuid(uid_t uid);

41004 **DESCRIPTION**41005 If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and
41006 the saved set-user-ID of the calling process to *uid*.41007 If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the
41008 saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-
41009 user-ID shall remain unchanged.41010 The *setuid()* function shall not affect the supplementary group list in any way.41011 **RETURN VALUE**41012 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
41013 indicate the error.41014 **ERRORS**41015 The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more
41016 of the following are true:41017 [EINVAL] The value of the *uid* argument is invalid and not supported by the
41018 implementation.41019 [EPERM] The process does not have appropriate privileges and *uid* does not match the
41020 real user ID or the saved set-user-ID.41021 **EXAMPLES**

41022 None.

41023 **APPLICATION USAGE**

41024 None.

41025 **RATIONALE**41026 The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged
41027 processes reflect the behavior of different historical implementations. For portability, it is
41028 recommended that new non-privileged applications use the *seteuid()* and *setegid()* functions
41029 instead.41030 The saved set-user-ID capability allows a program to regain the effective user ID established at
41031 the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the
41032 effective group ID established at the last *exec* call. These capabilities are derived from System V.
41033 Without them, a program might have to run as superuser in order to perform the same
41034 functions, because superuser can write on the user's files. This is a problem because such a
41035 program can write on any user's files, and so must be carefully written to emulate the
41036 permissions of the calling process properly. In System V, these capabilities have traditionally
41037 been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The
41038 fact that the behavior of those functions was different for privileged processes made them
41039 difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently
41040 for privileged and unprivileged users. When the caller had the appropriate privilege, the
41041 function set the calling process' real user ID, effective user ID, and saved set-user ID on
41042 implementations that supported it. When the caller did not have the appropriate privilege, the
41043 function set only the effective user ID, subject to permission checks. The former use is generally
41044 needed for utilities like *login* and *su*, which are not conforming applications and thus outside the

scope of IEEE Std 1003.1-2001. These utilities wish to change the user ID irrevocably to a new value, generally that of an unprivileged user. The latter use is needed for conforming applications that are installed with the set-user-ID bit and need to perform operations using the real user ID.

IEEE Std 1003.1-2001 augments the latter functionality with a mandatory feature named `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user ID back and forth between the values of its *exec*-time real user ID and effective user ID. Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this feature to work properly when it happened to be executed with the (implementation-defined) appropriate privilege. Furthermore, the application did not even have a means to tell whether it had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by IEEE Std 1003.1-2001. However, there are implementors who have been reluctant to support it given the limitation described above.

The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which always sets both the real and effective user IDs, like *setuid()* in IEEE Std 1003.1-2001 for privileged users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in IEEE Std 1003.1-2001 for non-privileged users). This separation of functionality into distinct functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of switching the effective user ID back and forth via *setreuid()*, which permits reversing the real and effective user IDs. This model seems less desirable than the saved set-user-ID because the real user ID changes as a side effect. The current 4.4BSD includes saved effective IDs and uses them for *seteuid()* and *setegid()* as described above. The *setreuid()* and *setregid()* functions will be deprecated or removed.

The solution here is:

- Require that all implementations support the functionality of the saved set-user-ID, which is set by the *exec* functions and by privileged calls to *setuid()*.
- Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for non-privileged and privileged processes.

Historical systems have provided two mechanisms for a set-user-ID process to change its effective user ID to be the same as its real user ID in such a way that it could return to the original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID, or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs. The changes included in IEEE Std 1003.1-2001 provide a new mechanism using *seteuid()* in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid()* mechanism will have a saved set-user-ID for each process, and most of the behavior controlled by `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined. The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally be required to maintain compatibility with the older mechanisms previously supported by their systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS` behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the saved set-user-ID unmodified, the process would then have an effective user ID equal to the original real user ID, and both real and saved set-user-ID would be equal to the original effective user ID. In that state, the real user would be unable to kill the process, even though the effective user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS` was used. This is obviously not acceptable. The alternative choice, which is used in at least one implementation, is to change the saved set-user-ID to the effective user ID during most calls to *setreuid()*. The standard developers considered that alternative to be less correct than the

41094 retention of the old behavior of *kill()* in such systems. Current conforming applications shall
41095 accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to
41096 check the saved set-user-ID rather than the effective user ID.

41097 **FUTURE DIRECTIONS**

41098 None.

41099 **SEE ALSO**

41100 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, the
41101 Base Definitions volume of IEEE Std 1003.1-2001, **<sys/types.h>**, **<unistd.h>**

41102 **CHANGE HISTORY**

41103 First released in Issue 1. Derived from Issue 1 of the SVID.

41104 **Issue 6**

41105 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

41106 The following new requirements on POSIX implementations derive from alignment with the
41107 Single UNIX Specification:

- 41108 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
41109 required for conforming implementations of previous POSIX specifications, it was not
41110 required for UNIX applications.
- 41111 • The functionality associated with `_POSIX_SAVED_IDS` is now mandatory. This is a FIPS
41112 requirement.

41113 The following changes were made to align with the IEEE P1003.1a draft standard:

- 41114 • The effects of *setuid()* in processes without appropriate privileges are changed.
- 41115 • A requirement that the supplementary group list is not affected is added.

41116 NAME

41117 setutxent — reset the user accounting database to the first entry

41118 SYNOPSIS

41119 xSI #include <utmpx.h>

41120 void setutxent(void);

41121

41122 DESCRIPTION

41123 Refer to *endutxent()*.

41124 **NAME**

41125 setvbuf — assign buffering to a stream

41126 **SYNOPSIS**

41127 #include <stdio.h>

41128 int setvbuf(FILE *restrict *stream*, char *restrict *buf*, int *type*,
 41129 size_t *size*);

41130 **DESCRIPTION**

41131 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41132 conflict between the requirements described here and the ISO C standard is unintentional. This
 41133 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

41134 The *setvbuf()* function may be used after the stream pointed to by *stream* is associated with an
 41135 open file but before any other operation (other than an unsuccessful call to *setvbuf()*) is
 41136 performed on the stream. The argument *type* determines how *stream* shall be buffered, as
 41137 follows:

- 41138 • {_IOFBF} shall cause input/output to be fully buffered.
- 41139 • {_IOLBF} shall cause input/output to be line buffered.
- 41140 • {_IONBF} shall cause input/output to be unbuffered.

41141 If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by
 41142 *setvbuf()* and the argument *size* specifies the size of the array; otherwise, *size* may determine the
 41143 size of a buffer allocated by the *setvbuf()* function. The contents of the array at any time are
 41144 unspecified.

41145 For information about streams, see Section 2.5 (on page 34).

41146 **RETURN VALUE**

41147 Upon successful completion, *setvbuf()* shall return 0. Otherwise, it shall return a non-zero value
 41148 CX if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to
 41149 indicate the error.

41150 **ERRORS**

41151 The *setvbuf()* function may fail if:

41152 CX [EBADF] The file descriptor underlying *stream* is not valid.

41153 **EXAMPLES**

41154 None.

41155 **APPLICATION USAGE**

41156 A common source of error is allocating buffer space as an “automatic” variable in a code block,
 41157 and then failing to close the stream in the same block.

41158 With *setvbuf()*, allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are
 41159 used for the buffer area.

41160 Applications should note that many implementations only provide line buffering on input from
 41161 terminal devices.

41162 **RATIONALE**

41163 None.

41164 **FUTURE DIRECTIONS**

41165 None.

41166 **SEE ALSO**41167 Section 2.5 (on page 34), *fopen()*, *setbuf()*, the Base Definitions volume of IEEE Std 1003.1-2001,
41168 <**stdio.h**>41169 **CHANGE HISTORY**

41170 First released in Issue 1. Derived from Issue 1 of the SVID.

41171 **Issue 6**

41172 Extensions beyond the ISO C standard are marked.

41173 The *setvbuf()* prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

41174 **NAME**41175 shm_open — open a shared memory object (**REALTIME**)41176 **SYNOPSIS**

41177 SHM #include <sys/mman.h>

41178 int shm_open(const char *name, int oflag, mode_t mode);

41179

41180 **DESCRIPTION**

41181 The *shm_open()* function shall establish a connection between a shared memory object and a file
 41182 descriptor. It shall create an open file description that refers to the shared memory object and a
 41183 file descriptor that refers to that open file description. The file descriptor is used by other
 41184 functions to refer to that shared memory object. The *name* argument points to a string naming a
 41185 shared memory object. It is unspecified whether the name appears in the file system and is
 41186 visible to other functions that take pathnames as arguments. The *name* argument conforms to the
 41187 construction rules for a pathname. If *name* begins with the slash character, then processes calling
 41188 *shm_open()* with the same value of *name* refer to the same shared memory object, as long as that
 41189 name has not been removed. If *name* does not begin with the slash character, the effect is
 41190 implementation-defined. The interpretation of slash characters other than the leading slash
 41191 character in *name* is implementation-defined.

41192 If successful, *shm_open()* shall return a file descriptor for the shared memory object that is the
 41193 lowest numbered file descriptor not currently open for that process. The open file description is
 41194 new, and therefore the file descriptor does not share it with any other processes. It is unspecified
 41195 whether the file offset is set. The FD_CLOEXEC file descriptor flag associated with the new file
 41196 descriptor is set.

41197 The file status flags and file access modes of the open file description are according to the value
 41198 of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the
 41199 <fcntl.h> header. Applications specify exactly one of the first two values (access modes) below
 41200 in the value of *oflag*:

41201 O_RDONLY Open for read access only.

41202 O_RDWR Open for read or write access.

41203 Any combination of the remaining flags may be specified in the value of *oflag*:

41204 O_CREAT If the shared memory object exists, this flag has no effect, except as noted
 41205 under O_EXCL below. Otherwise, the shared memory object is created; the
 41206 user ID of the shared memory object shall be set to the effective user ID of the
 41207 process; the group ID of the shared memory object is set to a system default
 41208 group ID or to the effective group ID of the process. The permission bits of the
 41209 shared memory object shall be set to the value of the *mode* argument except
 41210 those set in the file mode creation mask of the process. When bits in *mode*
 41211 other than the file permission bits are set, the effect is unspecified. The *mode*
 41212 argument does not affect whether the shared memory object is opened for
 41213 reading, for writing, or for both. The shared memory object has a size of zero.

41214 O_EXCL If O_EXCL and O_CREAT are set, *shm_open()* fails if the shared memory
 41215 object exists. The check for the existence of the shared memory object and the
 41216 creation of the object if it does not exist is atomic with respect to other
 41217 processes executing *shm_open()* naming the same shared memory object with
 41218 O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the
 41219 result is undefined.

41220 O_TRUNC If the shared memory object exists, and it is successfully opened O_RDWR,
 41221 the object shall be truncated to zero length and the mode and owner shall be
 41222 unchanged by this function call. The result of using O_TRUNC with
 41223 O_RDONLY is undefined.

41224 When a shared memory object is created, the state of the shared memory object, including all
 41225 data associated with the shared memory object, persists until the shared memory object is
 41226 unlinked and all other references are gone. It is unspecified whether the name and shared
 41227 memory object state remain valid after a system reboot.

41228 RETURN VALUE

41229 Upon successful completion, the *shm_open()* function shall return a non-negative integer
 41230 representing the lowest numbered unused file descriptor. Otherwise, it shall return `-1` and set
 41231 *errno* to indicate the error.

41232 ERRORS

41233 The *shm_open()* function shall fail if:

41234 [EACCES] The shared memory object exists and the permissions specified by *oflag* are
 41235 denied, or the shared memory object does not exist and permission to create
 41236 the shared memory object is denied, or O_TRUNC is specified and write
 41237 permission is denied.

41238 [EEXIST] O_CREAT and O_EXCL are set and the named shared memory object already
 41239 exists.

41240 [EINTR] The *shm_open()* operation was interrupted by a signal.

41241 [EINVAL] The *shm_open()* operation is not supported for the given name.

41242 [EMFILE] Too many file descriptors are currently in use by this process.

41243 [ENAMETOOLONG]

41244 The length of the *name* argument exceeds {PATH_MAX} or a pathname
 41245 component is longer than {NAME_MAX}.

41246 [ENFILE] Too many shared memory objects are currently open in the system.

41247 [ENOENT] O_CREAT is not set and the named shared memory object does not exist.

41248 [ENOSPC] There is insufficient space for the creation of the new shared memory object.

41249 EXAMPLES

41250 None.

41251 APPLICATION USAGE

41252 None.

41253 RATIONALE

41254 When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a
 41255 descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared
 41256 Memory Objects option is supported, the *shm_open()* function shall obtain a descriptor to the
 41257 shared memory object to be mapped.

41258 There is ample precedent for having a file descriptor represent several types of objects. In the
 41259 POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory.
 41260 Many implementations simply have an operations vector, which is indexed by the file descriptor
 41261 type and does very different operations. Note that in some cases the file descriptor passed to
 41262 generic operations on file descriptors is returned by *open()* or *creat()* and in some cases returned
 41263 by alternate functions, such as *pipe()*. The latter technique is used by *shm_open()*.

Note that such shared memory objects can actually be implemented as mapped files. In both cases, the size can be set after the open using *ftruncate()*. The *shm_open()* function itself does not create a shared object of a specified size because this would duplicate an extant function that set the size of an object referenced by a file descriptor.

On implementations where memory objects are implemented using the existing file system, the *shm_open()* function may be implemented using a macro that invokes *open()*, and the *shm_unlink()* function may be implemented using a macro that invokes *unlink()*.

For implementations without a permanent file system, the definition of the name of the memory objects is allowed not to survive a system reboot. Note that this allows systems with a permanent file system to implement memory objects as data structures internal to the implementation as well.

On implementations that choose to implement memory objects using memory directly, a *shm_open()* followed by an *ftruncate()* and *close()* can be used to preallocate a shared memory area and to set the size of that preallocation. This may be necessary for systems without virtual memory hardware support in order to ensure that the memory is contiguous.

The set of valid open flags to *shm_open()* was restricted to *O_RDONLY*, *O_RDWR*, *O_CREAT*, and *O_TRUNC* because these could be easily implemented on most memory mapping systems. This volume of IEEE Std 1003.1-2001 is silent on the results if the implementation cannot supply the requested file access because of implementation-defined reasons, including hardware ones.

The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the implementation cannot complete a request.

[EACCES] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode because it conflicts with another requested mode. An example might be that an application desires to open a memory object two times, mapping different areas with different access modes. If the implementation cannot map a single area into a process space in two places, which would be required if different access modes were required for the two areas, then the implementation may inform the application at the time of the second open.

[ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode at all. An example would be that the hardware of the implementation cannot support write-only shared memory areas.

On all implementations, it may be desirable to restrict the location of the memory objects to specific file systems for performance (such as a RAM disk) or implementation-defined reasons (shared memory supported directly only on certain file systems). The *shm_open()* function may be used to enforce these restrictions. There are a number of methods available to the application to determine an appropriate name of the file or the location of an appropriate directory. One way is from the environment via *getenv()*. Another would be from a configuration file.

This volume of IEEE Std 1003.1-2001 specifies that memory objects have initial contents of zero when created. This is consistent with current behavior for both files and newly allocated memory. For those implementations that use physical memory, it would be possible that such implementations could simply use available memory and give it to the process uninitialized. This, however, is not consistent with standard behavior for the uninitialized data area, the stack, and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security reasons. Thus, initializing memory objects to zero is required.

41308 **FUTURE DIRECTIONS**

41309 None.

41310 **SEE ALSO**

41311 *close()*, *dup()*, *exec*, *fcntl()*, *mmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm_unlink()*, *umask()*, the Base
41312 Definitions volume of IEEE Std 1003.1-2001, **<fcntl.h>**, **<sys/mman.h>**

41313 **CHANGE HISTORY**

41314 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

41315 **Issue 6**41316 The *shm_open()* function is marked as part of the Shared Memory Objects option.

41317 The [ENOSYS] error condition has been removed as stubs need not be provided if an
41318 implementation does not support the Shared Memory Objects option.

41319 **NAME**41320 shm_unlink — remove a shared memory object (**REALTIME**)41321 **SYNOPSIS**

41322 SHM #include <sys/mman.h>

41323 int shm_unlink(const char *name);

41324

41325 **DESCRIPTION**

41326 The *shm_unlink()* function shall remove the name of the shared memory object named by the
 41327 string pointed to by *name*.

41328 If one or more references to the shared memory object exist when the object is unlinked, the
 41329 name shall be removed before *shm_unlink()* returns, but the removal of the memory object
 41330 contents shall be postponed until all open and map references to the shared memory object have
 41331 been removed.

41332 Even if the object continues to exist after the last *shm_unlink()*, reuse of the name shall
 41333 subsequently cause *shm_open()* to behave as if no shared memory object of this name exists (that
 41334 is, *shm_open()* will fail if O_CREAT is not set, or will create a new shared memory object if
 41335 O_CREAT is set).

41336 **RETURN VALUE**

41337 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be
 41338 returned and *errno* set to indicate the error. If -1 is returned, the named shared memory object
 41339 shall not be changed by this function call.

41340 **ERRORS**41341 The *shm_unlink()* function shall fail if:

41342 [EACCES] Permission is denied to unlink the named shared memory object.

41343 [ENAMETOOLONG]

41344 The length of the *name* argument exceeds {PATH_MAX} or a pathname
 41345 component is longer than {NAME_MAX}.

41346 [ENOENT] The named shared memory object does not exist.

41347 **EXAMPLES**

41348 None.

41349 **APPLICATION USAGE**

41350 Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual
 41351 fashion. Names of memory objects that were allocated with *shm_open()* are deleted with
 41352 *shm_unlink()*. Note that the actual memory object is not destroyed until the last close and
 41353 unmap on it have occurred if it was already in use.

41354 **RATIONALE**

41355 None.

41356 **FUTURE DIRECTIONS**

41357 None.

41358 **SEE ALSO**

41359 *close()*, *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm_open()*, the Base Definitions volume
 41360 of IEEE Std 1003.1-2001, <sys/mman.h>

CHANGE HISTORY

41361 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

41364 The *shm_unlink()* function is marked as part of the Shared Memory Objects option.

41365 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm_unlink()*
41366 will not attach to the old shared memory object.

41367 The [ENOSYS] error condition has been removed as stubs need not be provided if an
41368 implementation does not support the Shared Memory Objects option.

41369 **NAME**

41370 shmat — XSI shared memory attach operation

41371 **SYNOPSIS**

41372 xsi #include <sys/shm.h>

41373 void *shmat(int *shmid*, const void **shmaddr*, int *shmflg*);

41374

41375 **DESCRIPTION**

41376 The *shmat()* function operates on XSI shared memory (see the Base Definitions volume of
 41377 IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this
 41378 function interoperates with the realtime interprocess communication facilities defined in Section
 41379 2.8 (on page 41).

41380 The *shmat()* function attaches the shared memory segment associated with the shared memory
 41381 identifier specified by *shmid* to the address space of the calling process. The segment is attached
 41382 at the address specified by one of the following criteria:

- 41383 • If *shmaddr* is a null pointer, the segment is attached at the first available address as selected
 41384 by the system.
- 41385 • If *shmaddr* is not a null pointer and (*shmflg* & SHM_RND) is non-zero, the segment is attached
 41386 at the address given by (*shmaddr* - ((uintptr_t)*shmaddr* % SHMLBA)). The character '%' is the
 41387 C-language remainder operator.
- 41388 • If *shmaddr* is not a null pointer and (*shmflg* & SHM_RND) is 0, the segment is attached at the
 41389 address given by *shmaddr*.
- 41390 • The segment is attached for reading if (*shmflg* & SHM_RDONLY) is non-zero and the calling
 41391 process has read permission; otherwise, if it is 0 and the calling process has read and write
 41392 permission, the segment is attached for reading and writing.

41393 **RETURN VALUE**

41394 Upon successful completion, *shmat()* shall increment the value of *shm_nattch* in the data
 41395 structure associated with the shared memory ID of the attached shared memory segment and
 41396 return the segment's start address.

41397 Otherwise, the shared memory segment shall not be attached, *shmat()* shall return -1, and *errno*
 41398 shall be set to indicate the error.

41399 **ERRORS**41400 The *shmat()* function shall fail if:

41401 41402	[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 39).
41403 41404 41405 41406 41407	[EINVAL]	The value of <i>shmid</i> is not a valid shared memory identifier, the <i>shmaddr</i> is not a null pointer, and the value of (<i>shmaddr</i> - ((uintptr_t) <i>shmaddr</i> % SHMLBA)) is an illegal address for attaching shared memory; or the <i>shmaddr</i> is not a null pointer, (<i>shmflg</i> & SHM_RND) is 0, and the value of <i>shmaddr</i> is an illegal address for attaching shared memory.
41408 41409	[EMFILE]	The number of shared memory segments attached to the calling process would exceed the system-imposed limit.
41410 41411	[ENOMEM]	The available data space is not large enough to accommodate the shared memory segment.

41412 **EXAMPLES**

41413 None.

41414 **APPLICATION USAGE**

41415 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
41416 Application developers who need to use IPC should design their applications so that modules
41417 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the
41418 alternative interfaces.

41419 **RATIONALE**

41420 None.

41421 **FUTURE DIRECTIONS**

41422 None.

41423 **SEE ALSO**

41424 Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*, *shmget()*,
41425 *shm_open()*, *shm_unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<sys/shm.h>**

41426 **CHANGE HISTORY**

41427 First released in Issue 2. Derived from Issue 2 of the SVID.

41428 **Issue 5**

41429 Moved from SHARED MEMORY to BASE.

41430 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
41431 DIRECTIONS to a new APPLICATION USAGE section.

41432 **Issue 6**

41433 The Open Group Corrigendum U021/13 is applied.

41434 **NAME**

41435 shmctl — XSI shared memory control operations

41436 **SYNOPSIS**41437 xsi `#include <sys/shm.h>`41438 `int shmctl(int shmid, int cmd, struct shm_id_ds *buf);`

41439

41440 **DESCRIPTION**

41441 The *shmctl()* function operates on XSI shared memory (see the Base Definitions volume of
 41442 IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this
 41443 function interoperates with the realtime interprocess communication facilities defined in Section
 41444 2.8 (on page 41).

41445 The *shmctl()* function provides a variety of shared memory control operations as specified by
 41446 *cmd*. The following values for *cmd* are available:

41447 **IPC_STAT** Place the current value of each member of the **shm_id_ds** data structure
 41448 associated with *shmid* into the structure pointed to by *buf*. The contents of the
 41449 structure are defined in **<sys/shm.h>**.

41450 **IPC_SET** Set the value of the following members of the **shm_id_ds** data structure
 41451 associated with *shmid* to the corresponding value found in the structure
 41452 pointed to by *buf*:

41453 `shm_perm.uid`
 41454 `shm_perm.gid`
 41455 `shm_perm.mode` Low-order nine bits.

41456 **IPC_SET** can only be executed by a process that has an effective user ID equal
 41457 to either that of a process with appropriate privileges or to the value of
 41458 *shm_perm.cuid* or *shm_perm.uid* in the **shm_id_ds** data structure associated with
 41459 *shmid*.

41460 **IPC_RMID** Remove the shared memory identifier specified by *shmid* from the system and
 41461 destroy the shared memory segment and **shm_id_ds** data structure associated
 41462 with it. **IPC_RMID** can only be executed by a process that has an effective user
 41463 ID equal to either that of a process with appropriate privileges or to the value
 41464 of *shm_perm.cuid* or *shm_perm.uid* in the **shm_id_ds** data structure associated
 41465 with *shmid*.

41466 **RETURN VALUE**

41467 Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return `-1` and set *errno* to
 41468 indicate the error.

41469 **ERRORS**41470 The *shmctl()* function shall fail if:

41471 **[EACCES]** The argument *cmd* is equal to **IPC_STAT** and the calling process does not have
 41472 read permission; see Section 2.7 (on page 39).

41473 **[EINVAL]** The value of *shmid* is not a valid shared memory identifier, or the value of *cmd*
 41474 is not a valid command.

41475 **[EPERM]** The argument *cmd* is equal to **IPC_RMID** or **IPC_SET** and the effective user ID
 41476 of the calling process is not equal to that of a process with appropriate
 41477 privileges and it is not equal to the value of *shm_perm.cuid* or *shm_perm.uid* in
 41478 the data structure associated with *shmid*.

41479 The *shmctl()* function may fail if:

41480 [EOVERFLOW] The *cmd* argument is IPC_STAT and the *gid* or *uid* value is too large to be
41481 stored in the structure pointed to by the *buf* argument.

41482 **EXAMPLES**

41483 None.

41484 **APPLICATION USAGE**

41485 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
41486 Application developers who need to use IPC should design their applications so that modules
41487 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the
41488 alternative interfaces.

41489 **RATIONALE**

41490 None.

41491 **FUTURE DIRECTIONS**

41492 None.

41493 **SEE ALSO**

41494 Section 2.7 (on page 39), Section 2.8 (on page 41), *shmat()*, *shmdt()*, *shmget()*, *shm_open()*,
41495 *shm_unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/shm.h>

41496 **CHANGE HISTORY**

41497 First released in Issue 2. Derived from Issue 2 of the SVID.

41498 **Issue 5**

41499 Moved from SHARED MEMORY to BASE.

41500 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
41501 DIRECTIONS to a new APPLICATION USAGE section.

41502 **NAME**

41503 shmdt — XSI shared memory detach operation

41504 **SYNOPSIS**

41505 xsi #include <sys/shm.h>

41506 int shmdt(const void *shmaddr);

41507

41508 **DESCRIPTION**

41509 The *shmdt()* function operates on XSI shared memory (see the Base Definitions volume of
 41510 IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this
 41511 function interoperates with the realtime interprocess communication facilities defined in Section
 41512 2.8 (on page 41).

41513 The *shmdt()* function detaches the shared memory segment located at the address specified by
 41514 *shmaddr* from the address space of the calling process.

41515 **RETURN VALUE**

41516 Upon successful completion, *shmdt()* shall decrement the value of *shm_nattch* in the data
 41517 structure associated with the shared memory ID of the attached shared memory segment and
 41518 return 0.

41519 Otherwise, the shared memory segment shall not be detached, *shmdt()* shall return -1 , and *errno*
 41520 shall be set to indicate the error.

41521 **ERRORS**41522 The *shmdt()* function shall fail if:

41523 [EINVAL] The value of *shmaddr* is not the data segment start address of a shared
 41524 memory segment.

41525 **EXAMPLES**

41526 None.

41527 **APPLICATION USAGE**

41528 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
 41529 Application developers who need to use IPC should design their applications so that modules
 41530 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the
 41531 alternative interfaces.

41532 **RATIONALE**

41533 None.

41534 **FUTURE DIRECTIONS**

41535 None.

41536 **SEE ALSO**

41537 Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit()*, *fork()*, *shmat()*, *shmctl()*, *shmget()*,
 41538 *shm_open()*, *shm_unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/shm.h>

41539 **CHANGE HISTORY**

41540 First released in Issue 2. Derived from Issue 2 of the SVID.

41541 **Issue 5**

41542 Moved from SHARED MEMORY to BASE.

41543 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 41544 DIRECTIONS to a new APPLICATION USAGE section.

41545 **NAME**

41546 shmget — get an XSI shared memory segment

41547 **SYNOPSIS**41548 xsi `#include <sys/shm.h>`41549 `int shmget(key_t key, size_t size, int shmflg);`

41550

41551 **DESCRIPTION**

41552 The *shmget()* function operates on XSI shared memory (see the Base Definitions volume of
 41553 IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this
 41554 function interoperates with the realtime interprocess communication facilities defined in Section
 41555 2.8 (on page 41).

41556 The *shmget()* function shall return the shared memory identifier associated with *key*.

41557 A shared memory identifier, associated data structure, and shared memory segment of at least
 41558 *size* bytes (see <sys/shm.h>) are created for *key* if one of the following is true:

- 41559 • The argument *key* is equal to `IPC_PRIVATE`.
- 41560 • The argument *key* does not already have a shared memory identifier associated with it and
 41561 (*shmflg* & `IPC_CREAT`) is non-zero.

41562 Upon creation, the data structure associated with the new shared memory identifier shall be
 41563 initialized as follows:

- 41564 • The values of *shm_perm.cuid*, *shm_perm.uid*, *shm_perm.cgid*, and *shm_perm.gid* are set equal to
 41565 the effective user ID and effective group ID, respectively, of the calling process.
- 41566 • The low-order nine bits of *shm_perm.mode* are set equal to the low-order nine bits of *shmflg*.
- 41567 • The value of *shm_segsz* is set equal to the value of *size*.
- 41568 • The values of *shm_lpid*, *shm_nattch*, *shm_atime*, and *shm_dtime* are set equal to 0.
- 41569 • The value of *shm_ctime* is set equal to the current time.

41570 When the shared memory segment is created, it shall be initialized with all zero values.

41571 **RETURN VALUE**

41572 Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared
 41573 memory identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

41574 **ERRORS**

41575 The *shmget()* function shall fail if:

41576 41577 41578	[EACCES]	A shared memory identifier exists for <i>key</i> but operation permission as specified by the low-order nine bits of <i>shmflg</i> would not be granted; see Section 2.7 (on page 39).
41579 41580	[EEXIST]	A shared memory identifier exists for the argument <i>key</i> but (<i>shmflg</i> & <code>IPC_CREAT</code>) && (<i>shmflg</i> & <code>IPC_EXCL</code>) is non-zero.
41581 41582	[EINVAL]	A shared memory segment is to be created and the value of <i>size</i> is less than the system-imposed minimum or greater than the system-imposed maximum.
41583 41584 41585	[EINVAL]	No shared memory segment is to be created and a shared memory segment exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> and <i>size</i> is not 0.

41586 [ENOENT] A shared memory identifier does not exist for the argument *key* and (*shmflg*
 41587 &IPC_CREAT) is 0.

41588 [ENOMEM] A shared memory identifier and associated shared memory segment shall be
 41589 created, but the amount of available physical memory is not sufficient to fill
 41590 the request.

41591 [ENOSPC] A shared memory identifier is to be created, but the system-imposed limit on
 41592 the maximum number of allowed shared memory identifiers system-wide
 41593 would be exceeded.

EXAMPLES

41594
 41595 None.

APPLICATION USAGE

41597 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
 41598 Application developers who need to use IPC should design their applications so that modules
 41599 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the
 41600 alternative interfaces.

RATIONALE

41601
 41602 None.

FUTURE DIRECTIONS

41603
 41604 None.

SEE ALSO

41606 Section 2.7 (on page 39), Section 2.8 (on page 41), *shmat()*, *shmctl()*, *shmdt()*, *shm_open()*,
 41607 *shm_unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/shm.h>

CHANGE HISTORY

41608
 41609 First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

41610
 41611 Moved from SHARED MEMORY to BASE.

41612 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 41613 DIRECTIONS to a new APPLICATION USAGE section.

41614 **NAME**

41615 shutdown — shut down socket send and receive operations

41616 **SYNOPSIS**

41617 #include <sys/socket.h>

41618 int shutdown(int *socket*, int *how*);41619 **DESCRIPTION**41620 The *shutdown()* function shall cause all or part of a full-duplex connection on the socket
41621 associated with the file descriptor *socket* to be shut down.41622 The *shutdown()* function takes the following arguments:41623 *socket* Specifies the file descriptor of the socket.41624 *how* Specifies the type of shutdown. The values are as follows:

41625 SHUT_RD Disables further receive operations.

41626 SHUT_WR Disables further send operations.

41627 SHUT_RDWR Disables further send and receive operations.

41628 The *shutdown()* function disables subsequent send and/or receive operations on a socket,
41629 depending on the value of the *how* argument.41630 **RETURN VALUE**41631 Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno*
41632 set to indicate the error.41633 **ERRORS**41634 The *shutdown()* function shall fail if:41635 [EBADF] The *socket* argument is not a valid file descriptor.41636 [EINVAL] The *how* argument is invalid.

41637 [ENOTCONN] The socket is not connected.

41638 [ENOTSOCK] The *socket* argument does not refer to a socket.41639 The *shutdown()* function may fail if:

41640 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

41641 **EXAMPLES**

41642 None.

41643 **APPLICATION USAGE**

41644 None.

41645 **RATIONALE**

41646 None.

41647 **FUTURE DIRECTIONS**

41648 None.

41649 **SEE ALSO**41650 *getsockopt()*, *read()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendto()*, *setsockopt()*, *socket()*,
41651 *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>

41652 **CHANGE HISTORY**

41653 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

41654 **NAME**

41655 sigaction — examine and change a signal action

41656 **SYNOPSIS**41657 CX `#include <signal.h>`

```
41658 int sigaction(int sig, const struct sigaction *restrict act,
41659             struct sigaction *restrict oact);
```

41660

41661 **DESCRIPTION**

41662 The *sigaction()* function allows the calling process to examine and/or specify the action to be
 41663 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are
 41664 defined in **<signal.h>**.

41665 The structure **sigaction**, used to describe an action to be taken, is defined in the **<signal.h>**
 41666 header to include at least the following members:

41667

41668

41669

41670

41671

41672

41673

41674

41675

41676

41677

Member Type	Member Name	Description
void(*) (int)	<i>sa_handler</i>	Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
sigset_t	<i>sa_mask</i>	Additional set of signals to be blocked during execution of signal-catching function.
int	<i>sa_flags</i>	Special flags to affect behavior of signal.
void(*) (int, siginfo_t *, void *)	<i>sa_sigaction</i>	Pointer to a signal-catching function.

41678 The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application
 41679 shall not use both simultaneously.

41680 If the argument *act* is not a null pointer, it points to a structure specifying the action to be
 41681 associated with the specified signal. If the argument *oact* is not a null pointer, the action
 41682 previously associated with the signal is stored in the location pointed to by the argument *oact*. If
 41683 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to
 41684 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall
 41685 not be added to the signal mask using this mechanism; this restriction shall be enforced by the
 41686 system without causing an error to be indicated.

41687 If the SA_SIGINFO flag (see below) is cleared in the *sa_flags* field of the **sigaction** structure, the
 41688 XSI|RTS *sa_handler* field identifies the action to be associated with the specified signal. If the
 41689 SA_SIGINFO flag is set in the *sa_flags* field, and the implementation supports the Realtime
 41690 Signals Extension option or the XSI Extension option, the *sa_sigaction* field specifies a signal-
 41691 catching function. If the SA_SIGINFO bit is cleared and the *sa_handler* field specifies a signal-
 41692 catching function, or if the SA_SIGINFO bit is set, the *sa_mask* field identifies a set of signals that
 41693 shall be added to the signal mask of the thread before the signal-catching function is invoked. If
 41694 the *sa_handler* field specifies a signal-catching function, the *sa_mask* field identifies a set of
 41695 signals that shall be added to the process' signal mask before the signal-catching function is
 41696 invoked.

41697 The *sa_flags* field can be used to modify the behavior of the specified signal.

41698 The following flags, defined in the **<signal.h>** header, can be set in *sa_flags*:

41699 XSI	SA_NOCLDSTOP	Do not generate SIGCHLD when children stop or stopped children continue.
41700		
41701		If <i>sig</i> is SIGCHLD and the SA_NOCLDSTOP flag is not set in <i>sa_flags</i> , and the implementation supports the SIGCHLD signal, then a SIGCHLD signal shall be generated for the calling process whenever any of its child processes stop and a SIGCHLD signal may be generated for the calling process whenever any of its stopped child processes are continued. If <i>sig</i> is SIGCHLD and the SA_NOCLDSTOP flag is set in <i>sa_flags</i> , then the implementation shall not generate a SIGCHLD signal in this way.
41702		
41703		
41704 XSI		
41705		
41706		
41707		
41708 XSI	SA_ONSTACK	If set and an alternate signal stack has been declared with <i>sigaltstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.
41709		
41710		
41711 XSI	SA_RESETHAND	If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.
41712		
41713		Note: SIGILL and SIGTRAP cannot be automatically reset when delivered; the system silently enforces this restriction.
41714		
41715		Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.
41716		
41717		In addition, if this flag is set, <i>sigaction()</i> behaves as if the SA_NODEFER flag were also set.
41718		
41719 XSI	SA_RESTART	This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].
41720		
41721		
41722		
41723		
41724		
41725	SA_SIGINFO	If cleared and the signal is caught, the signal-catching function shall be entered as:
41726		
41727		<pre>void func(int signo);</pre>
41728		where <i>signo</i> is the only argument to the signal-catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal-catching function and the application shall not modify the <i>sa_sigaction</i> member.
41729		
41730		
41731		
41732 XSI RTS		If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:
41733		
41734		<pre>void func(int signo, siginfo_t *info, void *context);</pre>
41735		where two additional arguments are passed to the signal-catching function. The second argument shall point to an object of type siginfo_t explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type ucontext_t to refer to the receiving process' context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal-catching function and the application shall not modify the <i>sa_handler</i> member.
41736		
41737		
41738		
41739		
41740		
41741		
41742		
41743		The <i>si_signo</i> member contains the system-generated signal number.

41744 XSI		The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.
41745		
41746		
41747 XSI RTS		The <i>si_code</i> member contains a code identifying the cause of the signal.
41748 XSI		If the value of <i>si_code</i> is less than or equal to 0, then the signal was generated by a process and <i>si_pid</i> and <i>si_uid</i> , respectively, indicate the process ID and the real user ID of the sender. The <signal.h> header description contains information about the signal-specific contents of the elements of the siginfo_t type.
41749		
41750		
41751		
41752		
41753 XSI	SA_NOCLDWAIT	If set, and <i>sig</i> equals SIGCHLD, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and <i>wait()</i> , <i>waitid()</i> , and <i>waitpid()</i> shall fail and set <i>errno</i> to [ECHILD]. Otherwise, terminating child processes shall be transformed into zombie processes, unless SIGCHLD is set to SIG_IGN.
41754		
41755		
41756		
41757		
41758		
41759		
41760		
41761 XSI	SA_NODEFER	If set and <i>sig</i> is caught, <i>sig</i> shall not be added to the process' signal mask on entry to the signal handler unless it is included in <i>sa_mask</i> . Otherwise, <i>sig</i> shall always be added to the process' signal mask on entry to the signal handler.
41762		
41763		
41764		
41765		When a signal is caught by a signal-catching function installed by <i>sigaction()</i> , a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either <i>sigprocmask()</i> or <i>sigsuspend()</i> is made). This mask is formed by taking the union of the current signal mask and the value of the <i>sa_mask</i> for the signal being delivered unless SA_NODEFER or SA_RESETHAND is set, and then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.
41766		
41767		
41768 XSI		
41769		
41770		
41771		Once an action is installed for a specific signal, it shall remain installed until another action is explicitly requested (by another call to <i>sigaction()</i>), until the SA_RESETHAND flag causes resetting of the handler, or until one of the <i>exec</i> functions is called.
41772 XSI		
41773		
41774		If the previous action for <i>sig</i> had been established by <i>signal()</i> , the values of the fields returned in the structure pointed to by <i>oact</i> are unspecified, and in particular <i>oact->sa_handler</i> is not necessarily the same value passed to <i>signal()</i> . However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to <i>sigaction()</i> via the <i>act</i> argument, handling of the signal shall be as if the original call to <i>signal()</i> were repeated.
41775		
41776		
41777		
41778		
41779		If <i>sigaction()</i> fails, no new signal handler is installed.
41780		It is unspecified whether an attempt to set the action for a signal that cannot be caught or ignored to SIG_DFL is ignored or causes an error to be returned with <i>errno</i> set to [EINVAL].
41781		
41782		If SA_SIGINFO is not set in <i>sa_flags</i> , then the disposition of subsequent occurrences of <i>sig</i> when it is already pending is implementation-defined; the signal-catching function shall be invoked with a single argument. If the implementation supports the Realtime Signals Extension option, and if SA_SIGINFO is set in <i>sa_flags</i> , then subsequent occurrences of <i>sig</i> generated by <i>sigqueue()</i> or as a result of any signal-generating function that supports the specification of an application-defined value (when <i>sig</i> is already pending) shall be queued in FIFO order until delivered or accepted; the signal-catching function shall be invoked with three arguments. The application specified value is passed to the signal-catching function as the <i>si_value</i> member of the siginfo_t structure.
41783		
41784 RTS		
41785		
41786		
41787		
41788		
41789		
41790		

41791 The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the
 41792 same signal is unspecified.

41793 RETURN VALUE

41794 Upon successful completion, *sigaction()* shall return 0; otherwise, *-1* shall be returned, *errno* shall
 41795 be set to indicate the error, and no new signal-catching function shall be installed.

41796 ERRORS

41797 The *sigaction()* function shall fail if:

41798 [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a
 41799 signal that cannot be caught or ignore a signal that cannot be ignored.

41800 [ENOTSUP] The SA_SIGINFO bit flag is set in the *sa_flags* field of the **sigaction** structure,
 41801 and the implementation does not support either the Realtime Signals
 41802 Extension option, or the XSI Extension option.

41803 The *sigaction()* function may fail if:

41804 [EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be
 41805 caught or ignored (or both).

41806 EXAMPLES

41807 None.

41808 APPLICATION USAGE

41809 The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In
 41810 particular, *sigaction()* and *signal()* should not be used in the same process to control the same
 41811 signal. The behavior of reentrant functions, as defined in the DESCRIPTION, is as specified by
 41812 this volume of IEEE Std 1003.1-2001, regardless of invocation from a signal-catching function.
 41813 This is the only intended meaning of the statement that reentrant functions may be used in
 41814 signal-catching functions without restrictions. Applications must still consider all effects of such
 41815 functions on such things as data structures, files, and process state. In particular, application
 41816 writers need to consider the restrictions on interactions when interrupting *sleep()* and
 41817 interactions among multiple handles for a file description. The fact that any specific function is
 41818 listed as reentrant does not necessarily mean that invocation of that function from a signal-
 41819 catching function is recommended.

41820 In order to prevent errors arising from interrupting non-reentrant function calls, applications
 41821 should protect calls to these functions either by blocking the appropriate signals or through the
 41822 use of some programmatic semaphore (see *semget()*, *sem_init()*, *sem_open()*, and so on). Note in
 41823 particular that even the “safe” functions may modify *errno*; the signal-catching function, if not
 41824 executing as an independent thread, may want to save and restore its value. Naturally, the same
 41825 principles apply to the reentrancy of application routines and asynchronous data access. Note
 41826 that *longjmp()* and *siglongjmp()* are not in the list of reentrant functions. This is because the code
 41827 executing after *longjmp()* and *siglongjmp()* can call any unsafe functions with the same danger as
 41828 calling those unsafe functions directly from the signal handler. Applications that use *longjmp()*
 41829 and *siglongjmp()* from within signal handlers require rigorous protection in order to be portable.
 41830 Many of the other functions that are excluded from the list are traditionally implemented using
 41831 either *malloc()* or *free()* functions or the standard I/O library, both of which traditionally use
 41832 data structures in a non-reentrant manner. Since any combination of different functions using a
 41833 common data structure can cause reentrancy problems, this volume of IEEE Std 1003.1-2001
 41834 does not define the behavior when any unsafe function is called in a signal handler that
 41835 interrupts an unsafe function.

41836 If the signal occurs other than as the result of calling *abort()*, *kill()*, or *raise()*, the behavior is
 41837 undefined if the signal handler calls any function in the standard library other than one of the

functions listed in the table above or refers to any object with static storage duration other than by assigning a value to a static storage duration variable of type **volatile sig_atomic_t**. Furthermore, if such a call fails, the value of *errno* is unspecified.

Usually, the signal is executed on the stack that was in effect before the signal was delivered. An alternate stack may be specified to receive a subset of the signals being caught.

When the signal handler returns, the receiving process resumes execution at the point it was interrupted unless the signal handler makes other arrangements. If *longjmp()* or *_longjmp()* is used to leave the signal handler, then the signal mask must be explicitly restored by the process.

This volume of IEEE Std 1003.1-2001 defines the third argument of a signal handling function when SA_SIGINFO is set as a **void *** instead of a **ucontext_t ***, but without requiring type checking. New applications should explicitly cast the third argument of the signal handling function to **ucontext_t ***.

The BSD optional four argument signal handling function is not supported by this volume of IEEE Std 1003.1-2001. The BSD declaration would be:

```
void handler(int sig, int code, struct sigcontext *scp,
             char *addr);
```

where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer to the **sigcontext** structure, and *addr* is additional address information. Much the same information is available in the objects pointed to by the second argument of the signal handler specified when SA_SIGINFO is set.

RATIONALE

Although this volume of IEEE Std 1003.1-2001 requires that signals that cannot be ignored shall not be added to the signal mask when a signal-catching function is entered, there is no explicit requirement that subsequent calls to *sigaction()* reflect this in the information returned in the *oact* argument. In other words, if SIGKILL is included in the *sa_mask* field of *act*, it is unspecified whether or not a subsequent call to *sigaction()* returns with SIGKILL included in the *sa_mask* field of *oact*.

The SA_NOCLDSTOP flag, when supplied in the *act->sa_flags* parameter, allows overloading SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated child. Most conforming applications that catch SIGCHLD are expected to install signal-catching functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of interest, the use of the SA_NOCLDSTOP flag can prevent the overhead from invoking the signal-catching routine when they stop.

Some historical implementations also define other mechanisms for stopping processes, such as the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when processes stop due to this mechanism; however, that is beyond the scope of this volume of IEEE Std 1003.1-2001.

This volume of IEEE Std 1003.1-2001 requires that calls to *sigaction()* that supply a NULL *act* argument succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases and, in this respect, their behavior varies from *sigaction()*.

This volume of IEEE Std 1003.1-2001 requires that *sigaction()* properly save and restore a signal action set up by the ISO C standard *signal()* function. However, there is no guarantee that the reverse is true, nor could there be given the greater amount of information conveyed by the **sigaction** structure. Because of this, applications should avoid using both functions for the same signal in the same process. Since this cannot always be avoided in case of general-purpose

41885 library routines, they should always be implemented with *sigaction()*.

41886 It was intended that the *signal()* function should be implementable as a library routine using
41887 *sigaction()*.

41888 The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990
41889 standard to allow the application to request on a per-signal basis via an additional signal action
41890 flag that the extra parameters, including the application-defined signal value, if any, be passed
41891 to the signal-catching function.

41892 **FUTURE DIRECTIONS**

41893 None.

41894 **SEE ALSO**

41895 Section 2.4 (on page 28), *bsd_signal()*, *kill()*, *_longjmp()*, *longjmp()*, *raise()*, *semget()*, *sem_init()*,
41896 *sem_open()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *signal()*,
41897 *sigprocmask()*, *sigsuspend()*, *wait()*, *waitid()*, *waitpid()*, the Base Definitions volume of
41898 IEEE Std 1003.1-2001, <**signal.h**>, <**ucontext.h**>

41899 **CHANGE HISTORY**

41900 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

41901 **Issue 5**

41902 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX
41903 Threads Extension.

41904 In the DESCRIPTION, the second argument to *func* when SA_SIGINFO is set is no longer
41905 permitted to be NULL, and the description of permitted **siginfo_t** contents is expanded by
41906 reference to <**signal.h**>.

41907 Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP]
41908 error is deleted.

41909 **Issue 6**

41910 The Open Group Corrigendum U028/7 is applied. In the paragraph entitled “Signal Effects on
41911 Other Functions”, a reference to *sigpending()* is added.

41912 In the DESCRIPTION, the text “Signal Generation and Delivery”, “Signal Actions”, and “Signal
41913 Effects on Other Functions” are moved to a separate section of this volume of
41914 IEEE Std 1003.1-2001.

41915 Text describing functionality from the Realtime Signals option is marked.

41916 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 41917 • The [ENOTSUP] error condition is added.

41918 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

41919 The **restrict** keyword is added to the *sigaction()* prototype for alignment with the
41920 ISO/IEC 9899: 1999 standard.

41921 References to the *wait3()* function are removed.

41922 The SYNOPSIS is marked CX since the presence of this function in the <**signal.h**> header is an
41923 extension over the ISO C standard.

41924 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table
41925 describing the **sigaction** structure.

41926 **NAME**

41927 sigaddset — add a signal to a signal set

41928 **SYNOPSIS**41929 CX `#include <signal.h>`41930 `int sigaddset(sigset_t *set, int signo);`

41931

41932 **DESCRIPTION**41933 The *sigaddset()* function adds the individual signal specified by the *signo* to the signal set pointed
41934 to by *set*.41935 Applications shall call either *sigemptyset()* or *sigfillset()* at least once for each object of type
41936 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
41937 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
41938 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
41939 *sigwaitinfo()*, the results are undefined.41940 **RETURN VALUE**41941 Upon successful completion, *sigaddset()* shall return 0; otherwise, it shall return *-1* and set *errno*
41942 to indicate the error.41943 **ERRORS**41944 The *sigaddset()* function may fail if:41945 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.41946 **EXAMPLES**

41947 None.

41948 **APPLICATION USAGE**

41949 None.

41950 **RATIONALE**

41951 None.

41952 **FUTURE DIRECTIONS**

41953 None.

41954 **SEE ALSO**41955 Section 2.4 (on page 28), *sigaction()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,
41956 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001,
41957 **<signal.h>**41958 **CHANGE HISTORY**

41959 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

41960 **Issue 5**41961 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
41962 previous issues.41963 **Issue 6**

41964 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

41965 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
41966 extension over the ISO C standard.

41967 **NAME**

41968 sigaltstack — set and get signal alternate stack context

41969 **SYNOPSIS**41970 xSI `#include <signal.h>`41971 `int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);`

41972

41973 **DESCRIPTION**

41974 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack
 41975 for signal handlers for the current thread. Signals that have been explicitly declared to execute
 41976 on the alternate stack shall be delivered on the alternate stack.

41977 If *ss* is not a null pointer, it points to a **stack_t** structure that specifies the alternate signal stack
 41978 that shall take effect upon return from *sigaltstack()*. The *ss_flags* member specifies the new stack
 41979 state. If it is set to *SS_DISABLE*, the stack is disabled and *ss_sp* and *ss_size* are ignored.
 41980 Otherwise, the stack shall be enabled, and the *ss_sp* and *ss_size* members specify the new address
 41981 and size of the stack.

41982 The range of addresses starting at *ss_sp* up to but not including *ss_sp+ss_size* is available to the
 41983 implementation for use as the stack. This function makes no assumptions regarding which end
 41984 is the stack base and in which direction the stack grows as items are pushed.

41985 If *oss* is not a null pointer, on successful completion it shall point to a **stack_t** structure that
 41986 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss_sp*
 41987 and *ss_size* members specify the address and size of that stack. The *ss_flags* member specifies the
 41988 stack's state, and may contain one of the following values:

41989 **SS_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to
 41990 modify the alternate signal stack while the process is executing on it fail. This
 41991 flag shall not be modified by processes.

41992 **SS_DISABLE** The alternate signal stack is currently disabled.

41993 The value *SIGSTKSZ* is a system default specifying the number of bytes that would be used to
 41994 cover the usual case when manually allocating an alternate stack area. The value *MINSIGSTKSZ*
 41995 is defined to be the minimum stack size for a signal handler. In computing an alternate stack
 41996 size, a program should add that amount to its stack requirements to allow for the system
 41997 implementation overhead. The constants *SS_ONSTACK*, *SS_DISABLE*, *SIGSTKSZ*, and
 41998 *MINSIGSTKSZ* are defined in **<signal.h>**.

41999 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new
 42000 process image.

42001 In some implementations, a signal (whether or not indicated to execute on the alternate stack)
 42002 shall always execute on the alternate stack if it is delivered while another signal is being caught
 42003 using the alternate stack.

42004 Use of this function by library threads that are not bound to kernel-scheduled entities results in
 42005 undefined behavior.

42006 **RETURN VALUE**

42007 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return *-1* and set *errno*
 42008 to indicate the error.

42009 **ERRORS**42010 The *sigaltstack()* function shall fail if:

42011 [EINVAL] The *ss* argument is not a null pointer, and the *ss_flags* member pointed to by *ss*
 42012 contains flags other than *SS_DISABLE*.

42013 [ENOMEM] The size of the alternate stack area is less than *MINSIGSTKSZ*.

42014 [EPERM] An attempt was made to modify an active stack.

42015 **EXAMPLES**42016 **Allocating Memory for an Alternate Stack**

42017 The following example illustrates a method for allocating memory for an alternate stack.

```
42018 #include <signal.h>
42019 ...
42020 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
42021     /* Error return. */
42022     sigstk.ss_size = SIGSTKSZ;
42023     sigstk.ss_flags = 0;
42024     if (sigaltstack(&sigstk, (stack_t *)0) < 0)
42025         perror("sigaltstack");
```

42026 **APPLICATION USAGE**

42027 On some implementations, stack space is automatically extended as needed. On those
 42028 implementations, automatic extension is typically not available for an alternate stack. If the stack
 42029 overflows, the behavior is undefined.

42030 **RATIONALE**

42031 None.

42032 **FUTURE DIRECTIONS**

42033 None.

42034 **SEE ALSO**

42035 Section 2.4 (on page 28), *sigaction()*, *sigsetjmp()*, the Base Definitions volume of
 42036 IEEE Std 1003.1-2001, <**signal.h**>

42037 **CHANGE HISTORY**

42038 First released in Issue 4, Version 2.

42039 **Issue 5**

42040 Moved from X/OPEN UNIX extension to BASE.

42041 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in
 42042 previous issues.

42043 **Issue 6**

42044 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42045 The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the
 42046 ISO/IEC 9899:1999 standard.

42047 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence
 42048 to include “for the current thread”.

42049 NAME

42050 sigdelset — delete a signal from a signal set

42051 SYNOPSIS

42052 CX `#include <signal.h>`

42053 `int sigdelset(sigset_t *set, int signo);`

42054

42055 DESCRIPTION

42056 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set
42057 pointed to by *set*.

42058 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type
42059 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
42060 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
42061 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
42062 *sigwaitinfo()*, the results are undefined.

42063 RETURN VALUE

42064 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return *-1* and set *errno*
42065 to indicate the error.

42066 ERRORS

42067 The *sigdelset()* function may fail if:

42068	[EINVAL]	The <i>signo</i> argument is not a valid signal number, or is an unsupported signal
42069		number.

42070 EXAMPLES

42071 None.

42072 APPLICATION USAGE

42073 None.

42074 RATIONALE

42075 None.

42076 FUTURE DIRECTIONS

42077 None.

42078 SEE ALSO

42079 Section 2.4 (on page 28), *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,
42080 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001,
42081 **<signal.h>**

42082 CHANGE HISTORY

42083 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42084 Issue 5

42085 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
42086 previous issues.

42087 Issue 6

42088 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
42089 extension over the ISO C standard.

42090 **NAME**

42091 sigemptyset — initialize and empty a signal set

42092 **SYNOPSIS**42093 **CX** #include <signal.h>

42094 int sigemptyset(sigset_t *set);

42095

42096 **DESCRIPTION**42097 The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined
42098 in IEEE Std 1003.1-2001 are excluded.42099 **RETURN VALUE**42100 Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return *-1* and set
42101 *errno* to indicate the error.42102 **ERRORS**

42103 No errors are defined.

42104 **EXAMPLES**

42105 None.

42106 **APPLICATION USAGE**

42107 None.

42108 **RATIONALE**42109 The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or
42110 set) all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the
42111 structure, such as a version field, to permit binary-compatibility between releases where the size
42112 of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any
42113 other use of the signal set, even if such use is read-only (for example, as an argument to
42114 *sigpending()*). This function is not intended for dynamic allocation.42115 The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or
42116 exclude) all the signals defined in this volume of IEEE Std 1003.1-2001. Although it is outside the
42117 scope of this volume of IEEE Std 1003.1-2001 to place this requirement on signals that are
42118 implemented as extensions, it is recommended that implementation-defined signals also be
42119 affected by these functions. However, there may be a good reason for a particular signal not to
42120 be affected. For example, blocking or ignoring an implementation-defined signal may have
42121 undesirable side effects, whereas the default action for that signal is harmless. In such a case, it
42122 would be preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.42123 In early proposals there was no distinction between invalid and unsupported signals (the names
42124 of optional signals that were not supported by an implementation were not defined by that
42125 implementation). The [EINVAL] error was thus specified as a required error for invalid signals.
42126 With that distinction, it is not necessary to require implementations of these functions to
42127 determine whether an optional signal is actually supported, as that could have a significant
42128 performance impact for little value. The error could have been required for invalid signals and
42129 optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is
42130 optional in both cases.42131 **FUTURE DIRECTIONS**

42132 None.

42133 **SEE ALSO**

42134 Section 2.4 (on page 28), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigismember()*, *sigpending()*,
42135 *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

42136 **CHANGE HISTORY**

42137 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42138 **Issue 6**

42139 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
42140 extension over the ISO C standard.

42141 **NAME**

42142 sigfillset — initialize and fill a signal set

42143 **SYNOPSIS**42144 CX `#include <signal.h>`42145 `int sigfillset(sigset_t *set);`

42146

42147 **DESCRIPTION**42148 The *sigfillset()* function shall initialize the signal set pointed to by *set*, such that all signals
42149 defined in this volume of IEEE Std 1003.1-2001 are included.42150 **RETURN VALUE**42151 Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno*
42152 to indicate the error.42153 **ERRORS**

42154 No errors are defined.

42155 **EXAMPLES**

42156 None.

42157 **APPLICATION USAGE**

42158 None.

42159 **RATIONALE**42160 Refer to *sigemptyset()* (on page 1354).42161 **FUTURE DIRECTIONS**

42162 None.

42163 **SEE ALSO**42164 Section 2.4 (on page 28), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigismember()*,
42165 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001,
42166 **<signal.h>**42167 **CHANGE HISTORY**

42168 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42169 **Issue 6**42170 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
42171 extension over the ISO C standard.

42172 **NAME**

42173 sighold, sigignore, sigpause, sigrelse, sigset — signal management

42174 **SYNOPSIS**

```
42175 xsi      #include <signal.h>

42176      int sighold(int sig);
42177      int sigignore(int sig);
42178      int sigpause(int sig);
42179      int sigrelse(int sig);
42180      void (*sigset(int sig, void (*disp)(int)))(int);
42181
```

42182 **DESCRIPTION**

42183 Use of any of these functions is unspecified in a multi-threaded process.

42184 The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions provide simplified signal
42185 management.

42186 The *sigset()* function shall modify signal dispositions. The *sig* argument specifies the signal,
42187 which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's
42188 disposition, which may be SIG_DFL, SIG_IGN, or the address of a signal handler. If *sigset()* is
42189 used, and *disp* is the address of a signal handler, the system shall add *sig* to the calling process'
42190 signal mask before executing the signal handler; when the signal handler returns, the system
42191 shall restore the calling process' signal mask to its state prior to the delivery of the signal. In
42192 addition, if *sigset()* is used, and *disp* is equal to SIG_HOLD, *sig* shall be added to the calling
42193 process' signal mask and *sig*'s disposition shall remain unchanged. If *sigset()* is used, and *disp* is
42194 not equal to SIG_HOLD, *sig* shall be removed from the calling process' signal mask.

42195 The *sighold()* function shall add *sig* to the calling process' signal mask.42196 The *sigrelse()* function shall remove *sig* from the calling process' signal mask.42197 The *sigignore()* function shall set the disposition of *sig* to SIG_IGN.

42198 The *sigpause()* function shall remove *sig* from the calling process' signal mask and suspend the
42199 calling process until a signal is received. The *sigpause()* function shall restore the process' signal
42200 mask to its original state before returning.

42201 If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes
42202 shall not be transformed into zombie processes when they terminate. If the calling process
42203 subsequently waits for its children, and the process has no unwaited-for children that were
42204 transformed into zombie processes, it shall block until all of its children terminate, and *wait()*,
42205 *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

42206 **RETURN VALUE**

42207 Upon successful completion, *sigset()* shall return SIG_HOLD if the signal had been blocked and
42208 the signal's previous disposition if it had not been blocked. Otherwise, SIG_ERR shall be
42209 returned and *errno* set to indicate the error.

42210 The *sigpause()* function shall suspend execution of the thread until a signal is received,
42211 whereupon it shall return -1 and set *errno* to [EINTR].

42212 For all other functions, upon successful completion, 0 shall be returned. Otherwise, -1 shall be
42213 returned and *errno* set to indicate the error.

42214 **ERRORS**

42215 These functions shall fail if:

42216 [EINVAL] The *sig* argument is an illegal signal number.42217 The *sigset()* and *sigignore()* functions shall fail if:42218 [EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a
42219 signal that cannot be ignored.42220 **EXAMPLES**

42221 None.

42222 **APPLICATION USAGE**42223 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling
42224 signals; new applications should use *sigaction()* rather than *sigset()*.42225 The *sighold()* function, in conjunction with *sigrelse()* or *sigpause()*, may be used to establish
42226 critical regions of code that require the delivery of a signal to be temporarily deferred.42227 The *sigsuspend()* function should be used in preference to *sigpause()* for broader portability.42228 **RATIONALE**

42229 None.

42230 **FUTURE DIRECTIONS**

42231 None.

42232 **SEE ALSO**42233 Section 2.4 (on page 28), *exec*, *pause()*, *sigaction()*, *signal()*, *sigsuspend()*, *waitid()*, the Base
42234 Definitions volume of IEEE Std 1003.1-2001, <**signal.h**>42235 **CHANGE HISTORY**

42236 First released in Issue 4, Version 2.

42237 **Issue 5**

42238 Moved from X/OPEN UNIX extension to BASE.

42239 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the process'
42240 signal mask to its original state before returning.42241 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends
42242 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to
42243 [EINTR].42244 **Issue 6**

42245 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42246 References to the *wait3()* function are removed.

42247 The XSI functions are split out into their own reference page.

42248 NAME

42249 siginterrupt — allow signals to interrupt functions

42250 SYNOPSIS

42251 xSI #include <signal.h>

42252 int siginterrupt(int *sig*, int *flag*);

42253

42254 DESCRIPTION

42255 The *siginterrupt()* function shall change the restart behavior when a function is interrupted by
 42256 the specified signal. The function *siginterrupt(sig, flag)* has an effect as if implemented as:

```

42257 int siginterrupt(int sig, int flag) {
42258     int ret;
42259     struct sigaction act;

42260     (void) sigaction(sig, NULL, &act);
42261     if (flag)
42262         act.sa_flags &= ~SA_RESTART;
42263     else
42264         act.sa_flags |= SA_RESTART;
42265     ret = sigaction(sig, &act, NULL);
42266     return ret;
42267 }
```

42268 RETURN VALUE

42269 Upon successful completion, *siginterrupt()* shall return 0; otherwise, -1 shall be returned and
 42270 *errno* set to indicate the error.

42271 ERRORS

42272 The *siginterrupt()* function shall fail if:

42273 [EINVAL] The *sig* argument is not a valid signal number.

42274 EXAMPLES

42275 None.

42276 APPLICATION USAGE

42277 The *siginterrupt()* function supports programs written to historical system interfaces. A
 42278 conforming application, when being written or rewritten, should use *sigaction()* with the
 42279 SA_RESTART flag instead of *siginterrupt()*.

42280 RATIONALE

42281 None.

42282 FUTURE DIRECTIONS

42283 None.

42284 SEE ALSO

42285 Section 2.4 (on page 28), *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 42286 <signal.h>

42287 CHANGE HISTORY

42288 First released in Issue 4, Version 2.

42289 **Issue 5**

42290 Moved from X/OPEN UNIX extension to BASE.

42291 **Issue 6**42292 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/59 is applied, correcting the declaration in
42293 the sample implementation given in the DESCRIPTION. |

42294 **NAME**

42295 sigismember — test for a signal in a signal set

42296 **SYNOPSIS**42297 CX `#include <signal.h>`42298 `int sigismember(const sigset_t *set, int signo);`

42299

42300 **DESCRIPTION**42301 The *sigismember()* function shall test whether the signal specified by *signo* is a member of the set
42302 pointed to by *set*.42303 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type
42304 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
42305 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
42306 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
42307 *sigwaitinfo()*, the results are undefined.42308 **RETURN VALUE**42309 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of
42310 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.42311 **ERRORS**42312 The *sigismember()* function may fail if:42313 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal
42314 number.42315 **EXAMPLES**

42316 None.

42317 **APPLICATION USAGE**

42318 None.

42319 **RATIONALE**

42320 None.

42321 **FUTURE DIRECTIONS**

42322 None.

42323 **SEE ALSO**42324 Section 2.4 (on page 28), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigemptyset()*, *sigpending()*,
42325 *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**42326 **CHANGE HISTORY**

42327 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42328 **Issue 5**42329 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
42330 previous issues.42331 **Issue 6**42332 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
42333 extension over the ISO C standard.

42334 **NAME**

42335 siglongjmp — non-local goto with signal handling

42336 **SYNOPSIS**42337 CX `#include <setjmp.h>`42338 `void siglongjmp(sigjmp_buf env, int val);`

42339

42340 **DESCRIPTION**42341 The *siglongjmp()* function shall be equivalent to the *longjmp()* function, except as follows:

- 42342 • References to *setjmp()* shall be equivalent to *sigsetjmp()*.
- 42343 • The *siglongjmp()* function shall restore the saved signal mask if and only if the *env* argument
- 42344 was initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

42345 **RETURN VALUE**

42346 After *siglongjmp()* is completed, program execution shall continue as if the corresponding
 42347 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function
 42348 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.

42349 **ERRORS**

42350 No errors are defined.

42351 **EXAMPLES**

42352 None.

42353 **APPLICATION USAGE**

42354 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant
 42355 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

42356 **RATIONALE**

42357 None.

42358 **FUTURE DIRECTIONS**

42359 None.

42360 **SEE ALSO**

42361 *longjmp()*, *setjmp()*, *sigprocmask()*, *sigsetjmp()*, *sigsuspend()*, the Base Definitions volume of
 42362 IEEE Std 1003.1-2001, **<setjmp.h>**

42363 **CHANGE HISTORY**

42364 First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

42365 **Issue 5**

42366 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42367 **Issue 6**42368 The DESCRIPTION is rewritten in terms of *longjmp()*.

42369 The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an
 42370 extension over the ISO C standard.

42371 **NAME**

42372 signal — signal management

42373 **SYNOPSIS**

42374 #include <signal.h>

42375 void (*signal(int *sig*, void (**func*)(int)))(int);42376 **DESCRIPTION**

42377 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42378 conflict between the requirements described here and the ISO C standard is unintentional. This
 42379 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

42380 CX Use of this function is unspecified in a multi-threaded process.

42381 The *signal()* function chooses one of three ways in which receipt of the signal number *sig* is to be
 42382 subsequently handled. If the value of *func* is SIG_DFL, default handling for that signal shall
 42383 occur. If the value of *func* is SIG_IGN, the signal shall be ignored. Otherwise, the application
 42384 shall ensure that *func* points to a function to be called when that signal occurs. An invocation of
 42385 such a function because of a signal, or (recursively) of any further functions called by that
 42386 invocation (other than functions in the standard library), is called a “signal handler”.

42387 When a signal occurs, and *func* points to a function, it is implementation-defined whether the
 42388 equivalent of a:

42389 signal(*sig*, SIG_DFL);

42390 is executed or the implementation prevents some implementation-defined set of signals (at least
 42391 including *sig*) from occurring until the current signal handling has completed. (If the value of *sig*
 42392 is SIGILL, the implementation may alternatively define that no action is taken.) Next the
 42393 equivalent of:

42394 (**func*)(*sig*);

42395 is executed. If and when the function returns, if the value of *sig* was SIGFPE, SIGILL, or
 42396 SIGSEGV or any other implementation-defined value corresponding to a computational
 42397 exception, the behavior is undefined. Otherwise, the program shall resume execution at the
 42398 CX point it was interrupted. If the signal occurs as the result of calling the *abort()*, *raise()*, *kill()*,
 42399 *pthread_kill()*, or *sigqueue()* function, the signal handler shall not call the *raise()* function.

42400 CX If the signal occurs other than as the result of calling *abort()*, *raise()*, *kill()*, *pthread_kill()*, or
 42401 *sigqueue()*, the behavior is undefined if the signal handler refers to any object with static storage
 42402 duration other than by assigning a value to an object declared as volatile **sig_atomic_t**, or if the
 42403 signal handler calls any function in the standard library other than one of the functions listed in
 42404 Section 2.4 (on page 28). Furthermore, if such a call fails, the value of *errno* is unspecified.

42405 At program start-up, the equivalent of:

42406 signal(*sig*, SIG_IGN);

42407 is executed for some signals, and the equivalent of:

42408 signal(*sig*, SIG_DFL);

42409 CX is executed for all other signals (see *exec*).

42410 **RETURN VALUE**

42411 If the request can be honored, *signal()* shall return the value of *func* for the most recent call to
 42412 *signal()* for the specified signal *sig*. Otherwise, SIG_ERR shall be returned and a positive value
 42413 shall be stored in *errno*.

42414 ERRORS

42415 The *signal()* function shall fail if:

42416 CX	[EINVAL]	The <i>sig</i> argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.
42417		

42418 The *signal()* function may fail if:

42419 CX	[EINVAL]	An attempt was made to set the action to SIG_DFL for a signal that cannot be caught or ignored (or both).
42420		

42421 EXAMPLES

42422 None.

42423 APPLICATION USAGE

42424 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling signals; new applications should use *sigaction()* rather than *signal()*.

42425

42426 RATIONALE

42427 None.

42428 FUTURE DIRECTIONS

42429 None.

42430 SEE ALSO

42431 Section 2.4 (on page 28), *exec*, *pause()*, *sigaction()*, *sigsuspend()*, *waitid()*, the Base Definitions

42432 volume of IEEE Std 1003.1-2001, <**signal.h**>

42433 CHANGE HISTORY

42434 First released in Issue 1. Derived from Issue 1 of the SVID.

42435 Issue 5

42436 Moved from X/OPEN UNIX extension to BASE.

42437 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the process' signal mask to its original state before returning.

42438

42439 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to [EINTR].

42440

42441

42442 Issue 6

42443 Extensions beyond the ISO C standard are marked.

42444 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42445 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

42446 References to the *wait3()* function are removed.

42447 The *sighold()*, *sigignore()*, *sigrelse()*, and *sigset()* functions are split out onto their own reference

42448 page.

42449 **NAME**

42450 signbit — test sign

42451 **SYNOPSIS**

42452 #include <math.h>

42453 int signbit(real-floating x);

42454 **DESCRIPTION**

42455 cx The functionality described on this reference page is aligned with the ISO C standard. Any
42456 conflict between the requirements described here and the ISO C standard is unintentional. This
42457 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

42458 The *signbit()* macro shall determine whether the sign of its argument value is negative. NaNs,
42459 zeros, and infinities have a sign bit.

42460 **RETURN VALUE**

42461 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is
42462 negative.

42463 **ERRORS**

42464 No errors are defined.

42465 **EXAMPLES**

42466 None.

42467 **APPLICATION USAGE**

42468 None.

42469 **RATIONALE**

42470 None.

42471 **FUTURE DIRECTIONS**

42472 None.

42473 **SEE ALSO**

42474 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, the Base Definitions volume of
42475 IEEE Std 1003.1-2001, <math.h>

42476 **CHANGE HISTORY**

42477 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

42478 **NAME**

42479 sigpause — remove a signal from the signal mask and suspend the thread

42480 **SYNOPSIS**

42481 XSI #include <signal.h>

42482 int sigpause(int *sig*);

42483

42484 **DESCRIPTION**

42485 Refer to *sighold()*.

42486 **NAME**

42487 sigpending — examine pending signals

42488 **SYNOPSIS**42489 CX `#include <signal.h>`42490 `int sigpending(sigset_t *set);`

42491

42492 **DESCRIPTION**

42493 The *sigpending()* function shall store, in the location referenced by the *set* argument, the set of
42494 signals that are blocked from delivery to the calling thread and that are pending on the process
42495 or the calling thread.

42496 **RETURN VALUE**

42497 Upon successful completion, *sigpending()* shall return 0; otherwise, *-1* shall be returned and
42498 *errno* set to indicate the error.

42499 **ERRORS**

42500 No errors are defined.

42501 **EXAMPLES**

42502 None.

42503 **APPLICATION USAGE**

42504 None.

42505 **RATIONALE**

42506 None.

42507 **FUTURE DIRECTIONS**

42508 None.

42509 **SEE ALSO**

42510 *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigprocmask()*, the Base Definitions
42511 volume of IEEE Std 1003.1-2001, **<signal.h>**

42512 **CHANGE HISTORY**

42513 First released in Issue 3.

42514 **Issue 5**

42515 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42516 **Issue 6**

42517 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
42518 extension over the ISO C standard.

42519 NAME

42520 sigprocmask — examine and change blocked signals

42521 SYNOPSIS

42522 CX `#include <signal.h>`

42523 `int sigprocmask(int how, const sigset_t *restrict set,`
42524 `sigset_t *restrict oset);`

42525

42526 DESCRIPTION

42527 Refer to *pthread_sigmask()*.

42528 **NAME**42529 sigqueue — queue a signal to a process (**REALTIME**)42530 **SYNOPSIS**42531 RTS `#include <signal.h>`42532 `int sigqueue(pid_t pid, int signo, const union sigval value);`

42533

42534 **DESCRIPTION**

42535 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value
 42536 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking
 42537 is performed but no signal is actually sent. The null signal can be used to check the validity of
 42538 *pid*.

42539 The conditions required for a process to have permission to queue a signal to another process
 42540 are the same as for the *kill()* function.

42541 The *sigqueue()* function shall return immediately. If SA_SIGINFO is set for *signo* and if the
 42542 resources were available to queue the signal, the signal shall be queued and sent to the receiving
 42543 process. If SA_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving
 42544 process; it is unspecified whether *value* shall be sent to the receiving process as a result of this
 42545 call.

42546 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked
 42547 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*
 42548 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the
 42549 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the
 42550 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.
 42551 The selection order between realtime and non-realtime signals, or between multiple pending
 42552 non-realtime signals, is unspecified.

42553 **RETURN VALUE**

42554 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*
 42555 function shall return a value of zero. Otherwise, the function shall return a value of -1 and set
 42556 *errno* to indicate the error.

42557 **ERRORS**42558 The *sigqueue()* function shall fail if:

42559 [EAGAIN] No resources are available to queue the signal. The process has already
 42560 queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s), or
 42561 a system-wide resource limit has been exceeded.

42562 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

42563 [EPERM] The process does not have the appropriate privilege to send the signal to the
 42564 receiving process.

42565 [ESRCH] The process *pid* does not exist.

42566 **EXAMPLES**

42567 None.

42568 **APPLICATION USAGE**

42569 None.

42570 **RATIONALE**

42571 The *sigqueue()* function allows an application to queue a realtime signal to itself or to another
42572 process, specifying the application-defined value. This is common practice in realtime
42573 applications on existing realtime systems. It was felt that specifying another function in the
42574 *sig...* name space already carved out for signals was preferable to extending the interface to
42575 *kill()*.

42576 Such a function became necessary when the put/get event function of the message queues was
42577 removed. It should be noted that the *sigqueue()* function implies reduced performance in a
42578 security-conscious implementation as the access permissions between the sender and receiver
42579 have to be checked on each send when the *pid* is resolved into a target process. Such access
42580 checks were necessary only at message queue open in the previous interface.

42581 The standard developers required that *sigqueue()* have the same semantics with respect to the
42582 null signal as *kill()*, and that the same permission checking be used. But because of the difficulty
42583 of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the
42584 interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function
42585 queues a signal to a single process specified by the *pid* argument.

42586 The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An
42587 explicit limit on the number of queued signals that a process could send was introduced. While
42588 the limit is “per-sender”, this volume of IEEE Std 1003.1-2001 does not specify that the resources
42589 be part of the state of the sender. This would require either that the sender be maintained after
42590 exit until all signals that it had sent to other processes were handled or that all such signals that
42591 had not yet been acted upon be removed from the queue(s) of the receivers. This volume of
42592 IEEE Std 1003.1-2001 does not preclude this behavior, but an implementation that allocated
42593 queuing resources from a system-wide pool (with per-sender limits) and that leaves queued
42594 signals pending after the sender exits is also permitted.

42595 **FUTURE DIRECTIONS**

42596 None.

42597 **SEE ALSO**

42598 Section 2.8.1 (on page 41), the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>

42599 **CHANGE HISTORY**

42600 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
42601 POSIX Threads Extension.

42602 **Issue 6**42603 The *sigqueue()* function is marked as part of the Realtime Signals Extension option.

42604 The [ENOSYS] error condition has been removed as stubs need not be provided if an
42605 implementation does not support the Realtime Signals Extension option.

42606 **NAME**

42607 sigrelse, sigset — signal management

42608 **SYNOPSIS**42609 XSI `#include <signal.h>`42610 `int sigrelse(int sig);`42611 `void (*sigset(int sig, void (*disp)(int)))(int);`

42612

42613 **DESCRIPTION**42614 Refer to *sighold()*.

42615 **NAME**

42616 sigsetjmp — set jump point for a non-local goto

42617 **SYNOPSIS**42618 CX `#include <setjmp.h>`42619 `int sigsetjmp(sigjmp_buf env, int savemask);`

42620

42621 **DESCRIPTION**42622 The *sigsetjmp()* function shall be equivalent to the *setjmp()* function, except as follows:

- 42623 • References to *setjmp()* are equivalent to *sigsetjmp()*.
- 42624 • References to *longjmp()* are equivalent to *siglongjmp()*.
- 42625 • If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal
- 42626 mask of the calling thread as part of the calling environment.

42627 **RETURN VALUE**

42628 If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from

42629 a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.

42630 **ERRORS**

42631 No errors are defined.

42632 **EXAMPLES**

42633 None.

42634 **APPLICATION USAGE**

42635 The distinction between *setjmp()*/*longjmp()* and *sigsetjmp()*/*siglongjmp()* is only significant for

42636 programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

42637 Note that since this function is defined in terms of *setjmp()*, if *savemask* is zero, it is unspecified

42638 whether the signal mask is saved.

42639 **RATIONALE**

42640 The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to

42641 permit implementors to recognize the name in the compiler and not implement an actual

42642 function. These same restrictions apply to the *sigsetjmp()* macro.

42643 There are processors that cannot easily support these calls, but this was not considered a

42644 sufficient reason to exclude them.

42645 4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *_setjmp()* and

42646 *_longjmp()* that, together with *setjmp()* and *longjmp()*, provide the same functionality as

42647 *sigsetjmp()* and *siglongjmp()*. On those systems, *setjmp()* and *longjmp()* save and restore signal

42648 masks, while *_setjmp()* and *_longjmp()* do not. On System V Release 3 and in corresponding

42649 issues of the SVID, *setjmp()* and *longjmp()* are explicitly defined not to save and restore signal

42650 masks. In order to permit existing practice in both cases, the relation of *setjmp()* and *longjmp()* to

42651 signal masks is not specified, and a new set of functions is defined instead.

42652 The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching

42653 *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts

42654 saved by other threads would be at best a questionable practice and were not considered worthy

42655 of standardization.

42656 **FUTURE DIRECTIONS**

42657 None.

42658 **SEE ALSO**

42659 *siglongjmp()*, *signal()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of
42660 IEEE Std 1003.1-2001, **<setjmp.h>**

42661 **CHANGE HISTORY**

42662 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42663 **Issue 5**

42664 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42665 **Issue 6**42666 The DESCRIPTION is reworded in terms of *setjmp()*.

42667 The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an
42668 extension over the ISO C standard.

42669 **NAME**

42670 sigsuspend — wait for a signal

42671 **SYNOPSIS**42672 CX `#include <signal.h>`42673 `int sigsuspend(const sigset_t *sigmask);`

42674

42675 **DESCRIPTION**

42676 The *sigsuspend()* function shall replace the current signal mask of the calling thread with the set
 42677 of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose
 42678 action is either to execute a signal-catching function or to terminate the process. This shall not
 42679 cause any other signals that may have been pending on the process to become pending on the
 42680 thread.

42681 If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to
 42682 execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching
 42683 function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()*
 42684 call.

42685 It is not possible to block signals that cannot be ignored. This is enforced by the system without
 42686 causing an error to be indicated.

42687 **RETURN VALUE**

42688 Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion
 42689 return value. If a return occurs, `-1` shall be returned and *errno* set to indicate the error.

42690 **ERRORS**42691 The *sigsuspend()* function shall fail if:

42692 [EINTR] A signal is caught by the calling process and control is returned from the
 42693 signal-catching function.

42694 **EXAMPLES**

42695 None.

42696 **APPLICATION USAGE**

42697 Normally, at the beginning of a critical code section, a specified set of signals is blocked using
 42698 the *sigprocmask()* function. When the thread has completed the critical section and needs to wait
 42699 for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was
 42700 returned by the *sigprocmask()* call.

42701 **RATIONALE**

42702 None.

42703 **FUTURE DIRECTIONS**

42704 None.

42705 **SEE ALSO**

42706 Section 2.4 (on page 28), *pause()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, the
 42707 Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

42708 **CHANGE HISTORY**

42709 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42710 **Issue 5**

42711 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42712 **Issue 6**

42713 The text in the RETURN VALUE section has been changed from “suspends process execution”
42714 to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

42715 Text in the APPLICATION USAGE section has been replaced.

42716 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an
42717 extension over the ISO C standard.

42718 **NAME**42719 sigtimedwait, sigwaitinfo — wait for queued signals (**REALTIME**)42720 **SYNOPSIS**42721 RTS

```
#include <signal.h>
```

```
42722 int sigtimedwait(const sigset_t *restrict set,
42723                 siginfo_t *restrict info,
42724                 const struct timespec *restrict timeout);
42725 int sigwaitinfo(const sigset_t *restrict set,
42726                 siginfo_t *restrict info);
42727
```

42728 **DESCRIPTION**

42729 The *sigtimedwait()* function shall be equivalent to *sigwaitinfo()* except that if none of the signals
 42730 specified by *set* are pending, *sigtimedwait()* shall wait for the time interval specified in the
 42731 **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is
 42732 zero-valued and if none of the signals specified by *set* are pending, then *sigtimedwait()* shall
 42733 MON return immediately with an error. If *timeout* is the NULL pointer, the behavior is unspecified. If
 42734 the Monotonic Clock option is supported, the CLOCK_MONOTONIC clock shall be used to
 42735 measure the time interval specified by the *timeout* argument.

42736 The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of
 42737 multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the
 42738 lowest numbered one. The selection order between realtime and non-realtime signals, or
 42739 between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at
 42740 the time of the call, the calling thread shall be suspended until one or more signals in *set* become
 42741 pending or until it is interrupted by an unblocked, caught signal.

42742 The *sigwaitinfo()* function shall be equivalent to the *sigwait()* function if the *info* argument is
 42743 NULL. If the *info* argument is non-NULL, the *sigwaitinfo()* function shall be equivalent to
 42744 *sigwait()*, except that the selected signal number shall be stored in the *si_signo* member, and the
 42745 cause of the signal shall be stored in the *si_code* member. If any value is queued to the selected
 42746 signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the
 42747 value shall be stored in the *si_value* member of *info*. The system resource used to queue the
 42748 signal shall be released and returned to the system for other use. If no value is queued, the
 42749 content of the *si_value* member is undefined. If no further signals are queued for the selected
 42750 signal, the pending indication for that signal shall be reset.

42751 **RETURN VALUE**

42752 Upon successful completion (that is, one of the signals specified by *set* is pending or is
 42753 generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise,
 42754 the function shall return a value of -1 and set *errno* to indicate the error.

42755 **ERRORS**42756 The *sigtimedwait()* function shall fail if:42757 [EAGAIN] No signal specified by *set* was generated within the specified timeout period.42758 The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

42759 [EINTR] The wait was interrupted by an unblocked, caught signal. It shall be
 42760 documented in system documentation whether this error causes these
 42761 functions to fail.

42762 The *sigtimedwait()* function may also fail if:

42763 [EINVAL] The *timeout* argument specified a *tv_nsec* value less than zero or greater than
42764 or equal to 1 000 million.

42765 An implementation only checks for this error if no signal is pending in *set* and it is necessary to
42766 wait.

42767 EXAMPLES

42768 None.

42769 APPLICATION USAGE

42770 The *sigtimedwait()* function times out and returns an [EAGAIN] error. Application writers
42771 should note that this is inconsistent with other functions such as *pthread_cond_timedwait()* that
42772 return [ETIMEDOUT].

42773 RATIONALE

42774 Existing programming practice on realtime systems uses the ability to pause waiting for a
42775 selected set of events and handle the first event that occurs in-line instead of in a signal-handling
42776 function. This allows applications to be written in an event-directed style similar to a state
42777 machine. This style of programming is useful for largescale transaction processing in which the
42778 overall throughput of an application and the ability to clearly track states are more important
42779 than the ability to minimize the response time of individual event handling.

42780 It is possible to construct a signal-waiting macro function out of the realtime signal function
42781 mechanism defined in this volume of IEEE Std 1003.1-2001. However, such a macro has to
42782 include the definition of a generalized handler for all signals to be waited on. A significant
42783 portion of the overhead of handler processing can be avoided if the signal-waiting function is
42784 provided by the kernel. This volume of IEEE Std 1003.1-2001 therefore provides two signal-
42785 waiting functions—one that waits indefinitely and one with a timeout—as part of the overall
42786 realtime signal function specification.

42787 The specification of a function with a timeout allows an application to be written that can be
42788 broken out of a wait after a set period of time if no event has occurred. It was argued that setting
42789 a timer event before the wait and recognizing the timer event in the wait would also implement
42790 the same functionality, but at a lower performance level. Because of the performance
42791 degradation associated with the user-level specification of a timer event and the subsequent
42792 cancellation of that timer event after the wait completes for a valid event, and the complexity
42793 associated with handling potential race conditions associated with the user-level method, the
42794 separate function has been included.

42795 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*
42796 function defined by this volume of IEEE Std 1003.1-2001. The only difference is that *sigwaitinfo()*
42797 returns the queued signal value in the *value* argument. The return of the queued value is
42798 required so that applications can differentiate between multiple events queued to the same
42799 signal number.

42800 The two distinct functions are being maintained because some implementations may choose to
42801 implement the POSIX Threads Extension functions and not implement the queued signals
42802 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*
42803 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a
42804 macro on *sigwaitinfo()*.

42805 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns
42806 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed
42807 wait, and immediate return, and concerns regarding consistency with other functions where the
42808 conditional and timed waits were separate functions from the pure blocking function. The

42809 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a
42810 macro with a NULL pointer for *timeout*.

42811 The *sigwait* functions provide a synchronous mechanism for threads to wait for
42812 asynchronously-generated signals. One important question was how many threads that are
42813 suspended in a call to a *sigwait()* function for a signal should return from the call when the
42814 signal is sent. Four choices were considered:

- 42815 1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
- 42816 2. One or more threads return.
- 42817 3. All waiting threads return.
- 42818 4. Exactly one thread returns.

42819 Prohibiting multiple calls to *sigwait()* for the same signal was felt to be overly restrictive. The
42820 “one or more” behavior made implementation of conforming packages easy at the expense of
42821 forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait()* in
42822 application code in order to achieve predictable behavior. There was concern that the “all
42823 waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU
42824 resources by replicating the signals in the general case. Furthermore, no convincing examples
42825 could be presented that delivery to all was either simpler or more powerful than delivery to one.

42826 Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait*
42827 function for a signal should return when that signal occurs. This is not an onerous restriction as:

- 42828 • A multi-way signal wait can be built from the single-way wait.
- 42829 • Signals should only be handled by application-level code, as library routines cannot guess
42830 what the application wants to do with signals generated for the entire process.
- 42831 • Applications can thus arrange for a single thread to wait for any given signal and call any
42832 needed routines upon its arrival.

42833 In an application that is using signals for interprocess communication, signal processing is
42834 typically done in one place. Alternatively, if the signal is being caught so that process cleanup
42835 can be done, the signal handler thread can call separate process cleanup routines for each
42836 portion of the application. Since the application main line started each portion of the application,
42837 it is at the right abstraction level to tell each portion of the application to clean up.

42838 Certainly, there exist programming styles where it is logical to consider waiting for a single
42839 signal in multiple threads. A simple *sigwait_multiple()* routine can be constructed to achieve this
42840 goal. A possible implementation would be to have each *sigwait_multiple()* caller registered as
42841 having expressed interest in a set of signals. The caller then waits on a thread-specific condition
42842 variable. A single server thread calls a *sigwait()* function on the union of all registered signals.
42843 When the *sigwait()* function returns, the appropriate state is set and condition variables are
42844 broadcast. New *sigwait_multiple()* callers may cause the pending *sigwait()* call to be canceled
42845 and reissued in order to update the set of signals being waited for.

42846 FUTURE DIRECTIONS

42847 None.

42848 SEE ALSO

42849 Section 2.8.1 (on page 41), *pause()*, *pthread_sigmask()*, *sigaction()*, *sigpending()*, *sigsuspend()*,
42850 *sigwait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**signal.h**>, <**time.h**>

42851 **CHANGE HISTORY**

42852 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
42853 POSIX Threads Extension.

42854 **Issue 6**

42855 These functions are marked as part of the Realtime Signals Extension option.

42856 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo()* function
42857 has been corrected so that the second argument is of type **siginfo_t***.

42858 The [ENOSYS] error condition has been removed as stubs need not be provided if an
42859 implementation does not support the Realtime Signals Extension option.

42860 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
42861 CLOCK_MONOTONIC clock, if supported, is used to measure timeout intervals.

42862 The **restrict** keyword is added to the *sigtimedwait()* and *sigwaitinfo()* prototypes for alignment
42863 with the ISO / IEC 9899:1999 standard.

42864 **NAME**

42865 sigwait — wait for queued signals

42866 **SYNOPSIS**42867 CX `#include <signal.h>`42868 `int sigwait(const sigset_t *restrict set, int *restrict sig);`

42869

42870 **DESCRIPTION**

42871 The *sigwait()* function shall select a pending signal from *set*, atomically clear it from the system's
 42872 set of pending signals, and return that signal number in the location referenced by *sig*. If prior to
 42873 the call to *sigwait()* there are multiple pending instances of a single signal number, it is
 42874 implementation-defined whether upon successful return there are any remaining pending
 42875 signals for that signal number. If the implementation supports queued signals and there are
 42876 multiple signals queued for the signal number selected, the first such queued signal shall cause a
 42877 return from *sigwait()* and the remainder shall remain queued. If no signal in *set* is pending at the
 42878 time of the call, the thread shall be suspended until one or more becomes pending. The signals
 42879 defined by *set* shall have been blocked at the time of the call to *sigwait()*; otherwise, the behavior
 42880 is undefined. The effect of *sigwait()* on the signal actions for the signals in *set* is unspecified.

42881 If more than one thread is using *sigwait()* to wait for the same signal, no more than one of these
 42882 threads shall return from *sigwait()* with the signal number. Which thread returns from *sigwait()*
 42883 if more than a single thread is waiting is unspecified.

42884 RTS Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it
 42885 shall be the lowest numbered one. The selection order between realtime and non-realtime
 42886 signals, or between multiple pending non-realtime signals, is unspecified.

42887 **RETURN VALUE**

42888 Upon successful completion, *sigwait()* shall store the signal number of the received signal at the
 42889 location referenced by *sig* and return zero. Otherwise, an error number shall be returned to
 42890 indicate the error.

42891 **ERRORS**42892 The *sigwait()* function may fail if:

42893 [EINVAL] The *set* argument contains an invalid or unsupported signal number.

42894 **EXAMPLES**

42895 None.

42896 **APPLICATION USAGE**

42897 None.

42898 **RATIONALE**

42899 To provide a convenient way for a thread to wait for a signal, this volume of
 42900 IEEE Std 1003.1-2001 provides the *sigwait()* function. For most cases where a thread has to wait
 42901 for a signal, the *sigwait()* function should be quite convenient, efficient, and adequate.

42902 However, requests were made for a lower-level primitive than *sigwait()* and for semaphores that
 42903 could be used by threads. After some consideration, threads were allowed to use semaphores
 42904 and *sem_post()* was defined to be async-signal and async-cancel-safe.

42905 In summary, when it is necessary for code run in response to an asynchronous signal to notify a
 42906 thread, *sigwait()* should be used to handle the signal. Alternatively, if the implementation
 42907 provides semaphores, they also can be used, either following *sigwait()* or from within a signal
 42908 handling routine previously registered with *sigaction()*.

42909 **FUTURE DIRECTIONS**

42910 None.

42911 **SEE ALSO**

42912 Section 2.4 (on page 28), Section 2.8.1 (on page 41), *pause()*, *pthread_sigmask()*, *sigaction()*,
42913 *sigpending()*, *sigsuspend()*, *sigwaitinfo()*, the Base Definitions volume of IEEE Std 1003.1-2001,
42914 **<signal.h>**, **<time.h>**

42915 **CHANGE HISTORY**

42916 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
42917 POSIX Threads Extension.

42918 **Issue 6**

42919 The **restrict** keyword is added to the *sigwait()* prototype for alignment with the
42920 ISO / IEC 9899:1999 standard.

42921 **NAME**

42922 sigwaitinfo — wait for queued signals (**REALTIME**)

42923 **SYNOPSIS**

42924 RTS `#include <signal.h>`

42925 `int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);`

42926

42927 **DESCRIPTION**

42928 Refer to *sigtimedwait()*.

42929 **NAME**

42930 sin, sinf, sinl — sine function

42931 **SYNOPSIS**

42932 #include <math.h>

42933 double sin(double x);

42934 float sinf(float x);

42935 long double sinl(long double x);

42936 **DESCRIPTION**

42937 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42938 conflict between the requirements described here and the ISO C standard is unintentional. This
 42939 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

42940 These functions shall compute the sine of their argument *x*, measured in radians.

42941 An application wishing to check for error situations should set *errno* to zero and call
 42942 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 42943 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 42944 zero, an error has occurred.

42945 **RETURN VALUE**42946 Upon successful completion, these functions shall return the sine of *x*.42947 MX If *x* is NaN, a NaN shall be returned.42948 If *x* is ± 0 , *x* shall be returned.42949 If *x* is subnormal, a range error may occur and *x* should be returned.

42950 If *x* is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 42951 defined value shall be returned.

42952 **ERRORS**

42953 These functions shall fail if:

42954 MX **Domain Error** The *x* argument is $\pm \text{Inf}$.

42955 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 42956 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 42957 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 42958 shall be raised.

42959 These functions may fail if:

42960 MX **Range Error** The value of *x* is subnormal

42961 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 42962 then *errno* shall be set to [ERANGE]. If the integer expression
 42963 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 42964 floating-point exception shall be raised.

42965 **EXAMPLES**42966 **Taking the Sine of a 45-Degree Angle**

```
42967 #include <math.h>
42968 ...
42969 double radians = 45.0 * M_PI / 180;
42970 double result;
42971 ...
42972 result = sin(radians);
```

42973 **APPLICATION USAGE**

42974 These functions may lose accuracy when their argument is near a multiple of π or is far from 0.0.

42975 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
42976 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

42977 **RATIONALE**

42978 None.

42979 **FUTURE DIRECTIONS**

42980 None.

42981 **SEE ALSO**

42982 *asin()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
42983 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

42984 **CHANGE HISTORY**

42985 First released in Issue 1. Derived from Issue 1 of the SVID.

42986 **Issue 5**

42987 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
42988 in previous issues.

42989 **Issue 6**

42990 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

42991 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
42992 revised to align with the ISO/IEC 9899:1999 standard.

42993 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
42994 marked.

42995 **NAME**

42996 sinh, sinhf, sinhl — hyperbolic sine functions

42997 **SYNOPSIS**

42998 #include <math.h>

42999 double sinh(double x);

43000 float sinhf(float x);

43001 long double sinhl(long double x);

43002 **DESCRIPTION**

43003 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43004 conflict between the requirements described here and the ISO C standard is unintentional. This
 43005 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43006 These functions shall compute the hyperbolic sine of their argument *x*.

43007 An application wishing to check for error situations should set *errno* to zero and call
 43008 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 43009 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 43010 zero, an error has occurred.

43011 **RETURN VALUE**43012 Upon successful completion, these functions shall return the hyperbolic sine of *x*.

43013 If the result would cause an overflow, a range error shall occur and $\pm\text{HUGE\_VAL}$,
 43014 $\pm\text{HUGE\_VALF}$, and $\pm\text{HUGE\_VALL}$ (with the same sign as *x*) shall be returned as appropriate for
 43015 the type of the function.

43016 MX If *x* is NaN, a NaN shall be returned.43017 If *x* is ± 0 or $\pm\text{Inf}$, *x* shall be returned.43018 If *x* is subnormal, a range error may occur and *x* should be returned.43019 **ERRORS**

43020 These functions shall fail if:

43021 Range Error The result would cause an overflow.

43022 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 43023 then *errno* shall be set to [ERANGE]. If the integer expression
 43024 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 43025 floating-point exception shall be raised.

43026 These functions may fail if:

43027 MX Range Error The value *x* is subnormal.

43028 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 43029 then *errno* shall be set to [ERANGE]. If the integer expression
 43030 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 43031 floating-point exception shall be raised.

43032 **EXAMPLES**

43033 None.

43034 **APPLICATION USAGE**

43035 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
43036 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

43037 **RATIONALE**

43038 None.

43039 **FUTURE DIRECTIONS**

43040 None.

43041 **SEE ALSO**

43042 *asinh()*, *cosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tanh()*, the Base Definitions volume of
43043 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
43044 <math.h>

43045 **CHANGE HISTORY**

43046 First released in Issue 1. Derived from Issue 1 of the SVID.

43047 **Issue 5**

43048 The DESCRIPTION is updated to indicate how an application should check for an error. This
43049 text was previously published in the APPLICATION USAGE section.

43050 **Issue 6**43051 The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

43052 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
43053 revised to align with the ISO/IEC 9899:1999 standard.

43054 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
43055 marked.

43056 **NAME**

43057 sinl — sine function

43058 **SYNOPSIS**

43059 #include <math.h>

43060 long double sinl(long double x);

43061 **DESCRIPTION**43062 Refer to *sin()*.

43063 **NAME**

43064 sleep — suspend execution for an interval of time

43065 **SYNOPSIS**

43066 #include <unistd.h>

43067 unsigned sleep(unsigned *seconds*);43068 **DESCRIPTION**

43069 The *sleep()* function shall cause the calling thread to be suspended from execution until either
 43070 the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is
 43071 delivered to the calling thread and its action is to invoke a signal-catching function or to
 43072 terminate the process. The suspension time may be longer than requested due to the scheduling
 43073 of other activity by the system.

43074 If a SIGALRM signal is generated for the calling process during execution of *sleep()* and if the
 43075 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep()*
 43076 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also
 43077 unspecified whether it remains pending after *sleep()* returns or it is discarded.

43078 If a SIGALRM signal is generated for the calling process during execution of *sleep()*, except as a
 43079 result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from
 43080 delivery, it is unspecified whether that signal has any effect other than causing *sleep()* to return.

43081 If a signal-catching function interrupts *sleep()* and examines or changes either the time a
 43082 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or
 43083 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

43084 If a signal-catching function interrupts *sleep()* and calls *siglongjmp()* or *longjmp()* to restore an
 43085 environment saved prior to the *sleep()* call, the action associated with the SIGALRM signal and
 43086 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also
 43087 unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored
 43088 as part of the environment.

43089 XSI Interactions between *sleep()* and any of *setitimer()*, *ualarm()*, or *usleep()* are unspecified.

43090 **RETURN VALUE**

43091 If *sleep()* returns because the requested time has elapsed, the value returned shall be 0. If *sleep()*
 43092 returns due to delivery of a signal, the return value shall be the “unslept” amount (the requested
 43093 time minus the time actually slept) in seconds.

43094 **ERRORS**

43095 No errors are defined.

43096 **EXAMPLES**

43097 None.

43098 **APPLICATION USAGE**

43099 None.

43100 **RATIONALE**

43101 There are two general approaches to the implementation of the *sleep()* function. One is to use the
 43102 *alarm()* function to schedule a SIGALRM signal and then suspend the process waiting for that
 43103 signal. The other is to implement an independent facility. This volume of IEEE Std 1003.1-2001
 43104 permits either approach.

43105 In order to comply with the requirement that no primitive shall change a process attribute unless
 43106 explicitly described by this volume of IEEE Std 1003.1-2001, an implementation using SIGALRM
 43107 must carefully take into account any SIGALRM signal scheduled by previous *alarm()* calls, the

action previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened to that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()* would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The action and blocking for SIGALRM must be saved and restored.

Historical implementations often implement the SIGALRM-based version using *alarm()* and *pause()*. One such implementation is prone to infinite hangups, as described in *pause()*. Another such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid that window. That implementation introduces a different problem: when the SIGALRM signal interrupts a signal-catching function installed by the user to catch a different signal, the *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*, *alarm()*, and *sigsuspend()* can avoid these problems.

Despite all reasonable care, there are several very subtle, but detectable and unavoidable, differences between the two types of implementations. These are the cases mentioned in this volume of IEEE Std 1003.1-2001 where some other activity relating to SIGALRM takes place, and the results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of concern to most applications.

See also the discussion of the term *realtime* in *alarm()*.

Since *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early under *alarm()* applies to *sleep()* as well.

Application writers should note that the type of the argument *seconds* and the return value of *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the ISO C standard sets as 65535, and any application passing a larger value is restricting its portability. A different type was considered, but historical implementations, including those with a 16-bit **int** type, consistently use either **unsigned** or **int**.

Scheduling delays may cause the process to return from the *sleep()* function significantly after the requested time. In such cases, the return value should be set to zero, since the formula (requested time minus the time actually spent) yields a negative number and *sleep()* returns an **unsigned**.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), *getitimer()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*, *ualarm()*, *usleep()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

43148 NAME

43149 snprintf — print formatted output

43150 SYNOPSIS

43151 #include <stdio.h>

43152 int snprintf(char *restrict *s*, size_t *n*,

43153 const char *restrict *format*, ...);

43154 DESCRIPTION

43155 Refer to *fprintf()*.

43156 NAME

43157 socketmark — determine whether a socket is at the out-of-band mark

43158 SYNOPSIS

```
43159        #include <sys/socket.h>

43160        int socketmark(int s);
```

43161 DESCRIPTION

43162 The *socketmark()* function shall determine whether the socket specified by the descriptor *s* is at
 43163 the out-of-band data mark (see the System Interfaces volume of IEEE Std 1003.1-2001, Section
 43164 2.10.12, Socket Out-of-Band Data State). If the protocol for the socket supports out-of-band data
 43165 by marking the stream with an out-of-band data mark, the *socketmark()* function shall return 1
 43166 when all data preceding the mark has been read and the out-of-band data mark is the first
 43167 element in the receive queue. The *socketmark()* function shall not remove the mark from the
 43168 stream.

43169 RETURN VALUE

43170 Upon successful completion, the *socketmark()* function shall return a value indicating whether
 43171 the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data
 43172 preceding the mark has been read, the return value shall be 1; if there is no mark, or if data
 43173 precedes the mark in the receive queue, the *socketmark()* function shall return 0. Otherwise, it
 43174 shall return a value of -1 and set *errno* to indicate the error.

43175 ERRORS

43176 The *socketmark()* function shall fail if:

43177	[EBADF]	The <i>s</i> argument is not a valid file descriptor.
43178	[ENOTTY]	The <i>s</i> argument does not specify a descriptor for a socket.

43179 EXAMPLES

43180 None.

43181 APPLICATION USAGE

43182 The use of this function between receive operations allows an application to determine which
 43183 received data precedes the out-of-band data and which follows the out-of-band data.

43184 There is an inherent race condition in the use of this function. On an empty receive queue, the
 43185 current read of the location might well be at the “mark”, but the system has no way of knowing
 43186 that the next data segment that will arrive from the network will carry the mark, and
 43187 *socketmark()* will return false, and the next read operation will silently consume the mark.

43188 Hence, this function can only be used reliably when the application already knows that the out-
 43189 of-band data has been seen by the system or that it is known that there is data waiting to be read
 43190 at the socket (via SIGURG or *select()*). See Section 2.10.11 (on page 61), Section 2.10.12 (on page
 43191 61), Section 2.10.14 (on page 62), and *pselect()* for details.

43192 RATIONALE

43193 The *socketmark()* function replaces the historical SIOCATMARK command to *ioctl()* which
 43194 implemented the same functionality on many implementations. Using a wrapper function
 43195 follows the adopted conventions to avoid specifying commands to the *ioctl()* function, other
 43196 than those now included to support XSI STREAMS. The *socketmark()* function could be
 43197 implemented as follows:

```
43198        #include <sys/ioctl.h>

43199        int socketmark(int s)
43200        {
```

```
43201         int val;
43202         if (ioctl(s, SIOCATMARK, &val) == -1)
43203             return(-1);
43204         return(val);
43205     }
```

43206 The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of
43207 SIOCATMARK.

43208 **FUTURE DIRECTIONS**

43209 None.

43210 **SEE ALSO**

43211 *pselect()*, *recv()*, *recvmsg()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>

43212 **CHANGE HISTORY**

43213 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

43214 **NAME**

43215 socket — create an endpoint for communication

43216 **SYNOPSIS**

43217 #include <sys/socket.h>

43218 int socket(int *domain*, int *type*, int *protocol*);43219 **DESCRIPTION**

43220 The *socket()* function shall create an unbound socket in a communications domain, and return a
 43221 file descriptor that can be used in later function calls that operate on sockets.

43222 The *socket()* function takes the following arguments:

43223 *domain* Specifies the communications domain in which a socket is to be created.

43224 *type* Specifies the type of socket to be created.

43225 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*
 43226 of 0 causes *socket()* to use an unspecified default protocol appropriate for the
 43227 requested socket type.

43228 The *domain* argument specifies the address family used in the communications domain. The
 43229 address families supported by the system are implementation-defined.

43230 Symbolic constants that can be used for the domain argument are defined in the <sys/socket.h>
 43231 header.

43232 The *type* argument specifies the socket type, which determines the semantics of communication
 43233 over the socket. The following socket types are defined; implementations may specify additional
 43234 socket types:

43235 SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte
 43236 streams, and may provide a transmission mechanism for out-of-band
 43237 data.

43238 SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages
 43239 of fixed maximum length.

43240 SOCK_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode
 43241 transmission paths for records. A record can be sent using one or more
 43242 output operations and received using one or more input operations, but a
 43243 single operation never transfers part of more than one record. Record
 43244 boundaries are visible to the receiver via the MSG_EOR flag.

43245 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
 43246 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
 43247 be used. The protocols supported by the system are implementation-defined.

43248 The process may need to have appropriate privileges to use the *socket()* function or to create
 43249 some sockets.

43250 **RETURN VALUE**

43251 Upon successful completion, *socket()* shall return a non-negative integer, the socket file
 43252 descriptor. Otherwise, a value of -1 shall be returned and *errno* set to indicate the error.

43253 **ERRORS**

43254 The *socket()* function shall fail if:

43255 [EAFNOSUPPORT]

43256 The implementation does not support the specified address family.

- 43257 [EMFILE] No more file descriptors are available for this process.
- 43258 [ENFILE] No more file descriptors are available for the system.
- 43259 [EPROTONOSUPPORT]
 43260 The protocol is not supported by the address family, or the protocol is not
 43261 supported by the implementation.
- 43262 [EPROTOYPE] The socket type is not supported by the protocol.
- 43263 The *socket()* function may fail if:
- 43264 [EACCES] The process does not have appropriate privileges.
- 43265 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 43266 [ENOMEM] Insufficient memory was available to fulfill the request.

43267 EXAMPLES

43268 None.

43269 APPLICATION USAGE

43270 The documentation for specific address families specifies which protocols each address family
 43271 supports. The documentation for specific protocols specifies which socket types each protocol
 43272 supports.

43273 The application can determine whether an address family is supported by trying to create a
 43274 socket with *domain* set to the protocol in question.

43275 RATIONALE

43276 None.

43277 FUTURE DIRECTIONS

43278 None.

43279 SEE ALSO

43280 *accept()*, *bind()*, *connect()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*,
 43281 *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socketpair()*, the Base Definitions volume of
 43282 IEEE Std 1003.1-2001, <**netinet/in.h**>, <**sys/socket.h**>

43283 CHANGE HISTORY

43284 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

43285 **NAME**

43286 socketpair — create a pair of connected sockets

43287 **SYNOPSIS**

43288 #include <sys/socket.h>

```
43289 int socketpair(int domain, int type, int protocol,
43290               int socket_vector[2]);
```

43291 **DESCRIPTION**

43292 The *socketpair()* function shall create an unbound pair of connected sockets in a specified *domain*,
 43293 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two
 43294 sockets shall be identical. The file descriptors used in referencing the created sockets shall be
 43295 returned in *socket_vector*[0] and *socket_vector*[1].

43296 The *socketpair()* function takes the following arguments:

43297 *domain* Specifies the communications domain in which the sockets are to be created.

43298 *type* Specifies the type of sockets to be created.

43299 *protocol* Specifies a particular protocol to be used with the sockets. Specifying a
 43300 *protocol* of 0 causes *socketpair()* to use an unspecified default protocol
 43301 appropriate for the requested socket type.

43302 *socket_vector* Specifies a 2-integer array to hold the file descriptors of the created socket
 43303 pair.

43304 The *type* argument specifies the socket type, which determines the semantics of communications
 43305 over the socket. The following socket types are defined; implementations may specify additional
 43306 socket types:

43307 SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte
 43308 streams, and may provide a transmission mechanism for out-of-band
 43309 data.

43310 SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages
 43311 of fixed maximum length.

43312 SOCK_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode
 43313 transmission paths for records. A record can be sent using one or more
 43314 output operations and received using one or more input operations, but a
 43315 single operation never transfers part of more than one record. Record
 43316 boundaries are visible to the receiver via the MSG_EOR flag.

43317 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
 43318 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
 43319 be used. The protocols supported by the system are implementation-defined.

43320 The process may need to have appropriate privileges to use the *socketpair()* function or to create
 43321 some sockets.

43322 **RETURN VALUE**

43323 Upon successful completion, this function shall return 0; otherwise, -1 shall be returned and
 43324 *errno* set to indicate the error.

43325 **ERRORS**

43326 The *socketpair()* function shall fail if:

43327 [EAFNOSUPPORT]

43328 The implementation does not support the specified address family.

- 43329 [EMFILE] No more file descriptors are available for this process.
- 43330 [ENFILE] No more file descriptors are available for the system.
- 43331 [EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.
- 43332 [EPROTONOSUPPORT]
- 43333 The protocol is not supported by the address family, or the protocol is not
- 43334 supported by the implementation.
- 43335 [EPROTOTYPE] The socket type is not supported by the protocol.
- 43336 The *socketpair()* function may fail if:
- 43337 [EACCES] The process does not have appropriate privileges.
- 43338 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 43339 [ENOMEM] Insufficient memory was available to fulfill the request.

43340 EXAMPLES

43341 None.

43342 APPLICATION USAGE

43343 The documentation for specific address families specifies which protocols each address family
 43344 supports. The documentation for specific protocols specifies which socket types each protocol
 43345 supports.

43346 The *socketpair()* function is used primarily with UNIX domain sockets and need not be
 43347 supported for other domains.

43348 RATIONALE

43349 None.

43350 FUTURE DIRECTIONS

43351 None.

43352 SEE ALSO

43353 *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>

43354 CHANGE HISTORY

43355 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

43356 **NAME**

43357 sprintf — print formatted output

43358 **SYNOPSIS**

43359 #include <stdio.h>

43360 int sprintf(char *restrict *s*, const char *restrict *format*, ...);43361 **DESCRIPTION**43362 Refer to *fprintf()*.

43363 **NAME**

43364 sqrt, sqrtf, sqrtl — square root function

43365 **SYNOPSIS**

43366 #include <math.h>

43367 double sqrt(double x);

43368 float sqrtf(float x);

43369 long double sqrtl(long double x);

43370 **DESCRIPTION**

43371 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43372 conflict between the requirements described here and the ISO C standard is unintentional. This
 43373 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43374 These functions shall compute the square root of their argument x , $\sqrt{x}$.

43375 An application wishing to check for error situations should set *errno* to zero and call
 43376 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 43377 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 43378 zero, an error has occurred.

43379 **RETURN VALUE**43380 Upon successful completion, these functions shall return the square root of x .

43381 MX For finite values of $x < -0$, a domain error shall occur, and either a NaN (if supported), or an
 43382 implementation-defined value shall be returned.

43383 MX If x is NaN, a NaN shall be returned.

43384 If x is ± 0 or $+\text{Inf}$, x shall be returned.

43385 If x is $-\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 43386 defined value shall be returned.

43387 **ERRORS**

43388 These functions shall fail if:

43389 MX Domain Error The finite value of x is < -0 , or x is $-\text{Inf}$.

43390 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 43391 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 43392 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 43393 shall be raised.

43394 **EXAMPLES**43395 **Taking the Square Root of 9.0**

43396 #include <math.h>

43397 ...

43398 double x = 9.0;

43399 double result;

43400 ...

43401 result = sqrt(x);

43402 APPLICATION USAGE

43403 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
43404 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

43405 RATIONALE

43406 None.

43407 FUTURE DIRECTIONS

43408 None.

43409 SEE ALSO

43410 `feclearexcept()`, `fetestexcept()`, `isnan()`, the Base Definitions volume of IEEE Std 1003.1-2001,
43411 Section 4.18, Treatment of Error Conditions for Mathematical Functions, `<math.h>`, `<stdio.h>`

43412 CHANGE HISTORY

43413 First released in Issue 1. Derived from Issue 1 of the SVID.

43414 Issue 5

43415 The DESCRIPTION is updated to indicate how an application should check for an error. This
43416 text was previously published in the APPLICATION USAGE section.

43417 Issue 6

43418 The `sqrtf()` and `sqrtrl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

43419 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
43420 revised to align with the ISO/IEC 9899:1999 standard.

43421 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
43422 marked.

43423 **NAME**

43424 srand — pseudo-random number generator

43425 **SYNOPSIS**

43426 #include <stdlib.h>

43427 void srand(unsigned *seed*) ;43428 **DESCRIPTION**43429 Refer to *rand()*.

43430 **NAME**

43431 srand48 — seed the uniformly distributed double-precision pseudo-random number generator

43432 **SYNOPSIS**

43433 XSI #include <stdlib.h>

43434 void srand48(long *seedval*);

43435

43436 **DESCRIPTION**43437 Refer to *drand48()*.

43438 NAME

43439 srandom — seed pseudo-random number generator

43440 SYNOPSIS

43441 xSI #include <stdlib.h>

43442 void srandom(unsigned *seed*);

43443

43444 DESCRIPTION

43445 Refer to *initstate()*.

43446 **NAME**

43447 sscanf — convert formatted input

43448 **SYNOPSIS**

43449 #include <stdio.h>

43450 int sscanf(const char *restrict *s*, const char *restrict *format*, ...);43451 **DESCRIPTION**43452 Refer to *fscanf()*.

43453 **NAME**

43454 stat — get file status

43455 **SYNOPSIS**

43456 #include <sys/stat.h>

43457 int stat(const char *restrict *path*, struct stat *restrict *buf*);43458 **DESCRIPTION**

43459 The *stat()* function shall obtain information about the named file and write it to the area pointed to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or execute permission of the named file is not required. An implementation that provides additional or alternate file access control mechanisms may, under implementation-defined conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file specified by *path*.

43465 If the named file is a symbolic link, the *stat()* function shall continue pathname resolution using the contents of the symbolic link, and shall return information pertaining to the resulting file if the file exists.

43468 The *buf* argument is a pointer to a **stat** structure, as defined in the <sys/stat.h> header, into which information is placed concerning the file.

43470 The *stat()* function shall update any time-related fields (as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.7, File Times Update), before writing into the **stat** structure.

43473 Unless otherwise specified, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*, *st_ctime*, and *st_mtime* shall have meaningful values for all file types defined in this volume of IEEE Std 1003.1-2001. The value of the member *st_nlink* shall be set to the number of links to the file.

43477 **RETURN VALUE**

43478 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

43480 **ERRORS**43481 The *stat()* function shall fail if:

43482 [EACCES] Search permission is denied for a component of the path prefix.

43483 [EIO] An error occurred while reading from the file system.

43484 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

43486 [ENAMETOOLONG]

43487 The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

43489 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

43490 [ENOTDIR] A component of the path prefix is not a directory.

43491 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by *buf*.

43494 The `stat()` function may fail if:

43495 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
43496 resolution of the *path* argument.

43497 [ENAMETOOLONG]

43498 As a result of encountering a symbolic link in resolution of the *path* argument,
43499 the length of the substituted pathname string exceeded {PATH_MAX}.

43500 [EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.

43501 EXAMPLES

43502 Obtaining File Status Information

43503 The following example shows how to obtain file status information for a file named
43504 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
43505 #include <sys/types.h>
43506 #include <sys/stat.h>
43507 #include <fcntl.h>

43508 struct stat buffer;
43509 int      status;
43510 ...
43511 status = stat("/home/cnd/mod1", &buffer);
```

43512 Getting Directory Information

43513 The following example fragment gets status information for each entry in a directory. The call to
43514 the `stat()` function stores file information in the **stat** structure pointed to by *statbuf*. The lines
43515 that follow the `stat()` call format the fields in the **stat** structure for presentation to the user of the
43516 program.

```
43517 #include <sys/types.h>
43518 #include <sys/stat.h>
43519 #include <dirent.h>
43520 #include <pwd.h>
43521 #include <grp.h>
43522 #include <time.h>
43523 #include <locale.h>
43524 #include <langinfo.h>
43525 #include <stdio.h>
43526 #include <stdint.h>

43527 struct dirent *dp;
43528 struct stat   statbuf;
43529 struct passwd *pwd;
43530 struct group  *grp;
43531 struct tm     *tm;
43532 char          datestring[256];
43533 ...
43534 /* Loop through directory entries. */
43535 while ((dp = readdir(dir)) != NULL) {

43536     /* Get entry's information. */
43537     if (stat(dp->d_name, &statbuf) == -1)
```

```

43538         continue;

43539         /* Print out type, permissions, and number of links. */
43540         printf("%10.10s", sperm (statbuf.st_mode));
43541         printf("%4d", statbuf.st_nlink);

43542         /* Print out owner's name if it is found using getpwuid(). */
43543         if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
43544             printf(" %-8.8s", pwd->pw_name);
43545         else
43546             printf(" %-8d", statbuf.st_uid);

43547         /* Print out group name if it is found using getgrgid(). */
43548         if ((grp = getgrgid(statbuf.st_gid)) != NULL)
43549             printf(" %-8.8s", grp->gr_name);
43550         else
43551             printf(" %-8d", statbuf.st_gid);

43552         /* Print size of file. */
43553         printf(" %9jd", (intmax_t)statbuf.st_size);

43554         tm = localtime(&statbuf.st_mtime);

43555         /* Get localized date string. */
43556         strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);

43557         printf(" %s %s\n", datestring, dp->d_name);
43558     }

```

43559 APPLICATION USAGE

43560 None.

43561 RATIONALE

43562 The intent of the paragraph describing “additional or alternate file access control mechanisms”
 43563 is to allow a secure implementation where a process with a label that does not dominate the
 43564 file’s label cannot perform a *stat()* function. This is not related to read permission; a process with
 43565 a label that dominates the file’s label does not need read permission. An implementation that
 43566 supports write-up operations could fail *fstat()* function calls even though it has a valid file
 43567 descriptor open for writing.

43568 FUTURE DIRECTIONS

43569 None.

43570 SEE ALSO

43571 *fstat()*, *lstat()*, *readlink()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 43572 **<sys/stat.h>**, **<sys/types.h>**

43573 CHANGE HISTORY

43574 First released in Issue 1. Derived from Issue 1 of the SVID.

43575 Issue 5

43576 Large File Summit extensions are added.

43577 Issue 6

43578 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

43579 The following new requirements on POSIX implementations derive from alignment with the
 43580 Single UNIX Specification:

- 43581 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
43582 required for conforming implementations of previous POSIX specifications, it was not
43583 required for UNIX applications.
 - 43584 • The [EIO] mandatory error condition is added.
 - 43585 • The [ELOOP] mandatory error condition is added.
 - 43586 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
43587 files.
 - 43588 • The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are
43589 added.
- 43590 The following changes were made to align with the IEEE P1003.1a draft standard:
- 43591 • Details are added regarding the treatment of symbolic links.
 - 43592 • The [ELOOP] optional error condition is added.
- 43593 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 43594 The **restrict** keyword is added to the `stat()` prototype for alignment with the ISO/IEC 9899:1999
43595 standard.

43596 **NAME**

43597 statvfs — get file system information

43598 **SYNOPSIS**

43599 xSI #include <sys/statvfs.h>

43600 int statvfs(const char *restrict *path*, struct statvfs *restrict *buf*);

43601

43602 **DESCRIPTION**43603 Refer to *fstatvfs()*.

43604 **NAME**

43605 stderr, stdin, stdout — standard I/O streams

43606 **SYNOPSIS**

43607 #include <stdio.h>

43608 extern FILE *stderr, *stdin, *stdout;

43609 **DESCRIPTION**

43610 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43611 conflict between the requirements described here and the ISO C standard is unintentional. This
 43612 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43613 A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type
 43614 **FILE**. The *fopen()* function shall create certain descriptive data for a stream and return a pointer
 43615 to designate the stream in all further transactions. Normally, there are three open streams with
 43616 constant pointers declared in the <stdio.h> header and associated with the standard open files.

43617 At program start-up, three streams shall be predefined and need not be opened explicitly:
 43618 *standard input* (for reading conventional input), *standard output* (for writing conventional output),
 43619 and *standard error* (for writing diagnostic output). When opened, the standard error stream is not
 43620 fully buffered; the standard input and standard output streams are fully buffered if and only if
 43621 the stream can be determined not to refer to an interactive device.

43622 CX The following symbolic values in <unistd.h> define the file descriptors that shall be associated
 43623 with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

43624 STDIN_FILENO Standard input value, *stdin*. Its value is 0.

43625 STDOUT_FILENO Standard output value, *stdout*. Its value is 1.

43626 STDERR_FILENO Standard error value, *stderr*. Its value is 2.

43627 The *stderr* stream is expected to be open for reading and writing.

43628 **RETURN VALUE**

43629 None.

43630 **ERRORS**

43631 No errors are defined.

43632 **EXAMPLES**

43633 None.

43634 **APPLICATION USAGE**

43635 None.

43636 **RATIONALE**

43637 None.

43638 **FUTURE DIRECTIONS**

43639 None.

43640 **SEE ALSO**

43641 *fclose()*, *feof()*, *ferror()*, *fileno()*, *fopen()*, *fread()*, *fseek()*, *getc()*, *gets()*, *popen()*, *printf()*, *putc()*,
 43642 *puts()*, *read()*, *scanf()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vprintf()*, the Base Definitions
 43643 volume of IEEE Std 1003.1-2001, <stdio.h>, <unistd.h>

43644 **CHANGE HISTORY**

43645 First released in Issue 1.

43646 **Issue 6**

43647 Extensions beyond the ISO C standard are marked.

43648 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

43649 **NAME**

43650 strcasecmp, strncasecmp — case-insensitive string comparisons

43651 **SYNOPSIS**43652 XSI `#include <strings.h>`43653 `int strcasecmp(const char *s1, const char *s2);`43654 `int strncasecmp(const char *s1, const char *s2, size_t n);`

43655

43656 **DESCRIPTION**

43657 The *strcasecmp()* function shall compare, while ignoring differences in case, the string pointed to
43658 by *s1* to the string pointed to by *s2*. The *strncasecmp()* function shall compare, while ignoring
43659 differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to
43660 by *s2*.

43661 In the POSIX locale, *strcasecmp()* and *strncasecmp()* shall behave as if the strings had been
43662 converted to lowercase and then a byte comparison performed. The results are unspecified in
43663 other locales.

43664 **RETURN VALUE**

43665 Upon completion, *strcasecmp()* shall return an integer greater than, equal to, or less than 0, if the
43666 string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to
43667 by *s2*, respectively.

43668 Upon successful completion, *strncasecmp()* shall return an integer greater than, equal to, or less
43669 than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal
43670 to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

43671 **ERRORS**

43672 No errors are defined.

43673 **EXAMPLES**

43674 None.

43675 **APPLICATION USAGE**

43676 None.

43677 **RATIONALE**

43678 None.

43679 **FUTURE DIRECTIONS**

43680 None.

43681 **SEE ALSO**43682 The Base Definitions volume of IEEE Std 1003.1-2001, <**strings.h**>43683 **CHANGE HISTORY**

43684 First released in Issue 4, Version 2.

43685 **Issue 5**

43686 Moved from X/OPEN UNIX extension to BASE.

43687 **NAME**

43688 strcat — concatenate two strings

43689 **SYNOPSIS**

43690 #include <string.h>

43691 char *strcat(char *restrict s1, const char *restrict s2);

43692 **DESCRIPTION**

43693 cx The functionality described on this reference page is aligned with the ISO C standard. Any
43694 conflict between the requirements described here and the ISO C standard is unintentional. This
43695 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43696 The *strcat()* function shall append a copy of the string pointed to by *s2* (including the
43697 terminating null byte) to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites
43698 the null byte at the end of *s1*. If copying takes place between objects that overlap, the behavior is
43699 undefined.

43700 **RETURN VALUE**43701 The *strcat()* function shall return *s1*; no return value is reserved to indicate an error.43702 **ERRORS**

43703 No errors are defined.

43704 **EXAMPLES**

43705 None.

43706 **APPLICATION USAGE**

43707 This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3
43708 applications. Reliable error detection by this function was never guaranteed.

43709 **RATIONALE**

43710 None.

43711 **FUTURE DIRECTIONS**

43712 None.

43713 **SEE ALSO**43714 *strncat()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>43715 **CHANGE HISTORY**

43716 First released in Issue 1. Derived from Issue 1 of the SVID.

43717 **Issue 6**43718 The *strcat()* prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

43719 **NAME**

43720 strchr — string scanning operation

43721 **SYNOPSIS**

43722 #include <string.h>

43723 char *strchr(const char *s, int c);

43724 **DESCRIPTION**

43725 cx The functionality described on this reference page is aligned with the ISO C standard. Any
43726 conflict between the requirements described here and the ISO C standard is unintentional. This
43727 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43728 The *strchr()* function shall locate the first occurrence of *c* (converted to a **char**) in the string
43729 pointed to by *s*. The terminating null byte is considered to be part of the string.

43730 **RETURN VALUE**

43731 Upon completion, *strchr()* shall return a pointer to the byte, or a null pointer if the byte was not
43732 found.

43733 **ERRORS**

43734 No errors are defined.

43735 **EXAMPLES**

43736 None.

43737 **APPLICATION USAGE**

43738 None.

43739 **RATIONALE**

43740 None.

43741 **FUTURE DIRECTIONS**

43742 None.

43743 **SEE ALSO**43744 *strrchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>43745 **CHANGE HISTORY**

43746 First released in Issue 1. Derived from Issue 1 of the SVID.

43747 **Issue 6**

43748 Extensions beyond the ISO C standard are marked.

43749 **NAME**

43750 strcmp — compare two strings

43751 **SYNOPSIS**

43752 #include <string.h>

43753 int strcmp(const char *s1, const char *s2);

43754 **DESCRIPTION**

43755 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 43756 conflict between the requirements described here and the ISO C standard is unintentional. This
 43757 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43758 The *strcmp()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

43759 The sign of a non-zero return value shall be determined by the sign of the difference between the
 43760 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings
 43761 being compared.

43762 **RETURN VALUE**

43763 Upon completion, *strcmp()* shall return an integer greater than, equal to, or less than 0, if the
 43764 string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*,
 43765 respectively.

43766 **ERRORS**

43767 No errors are defined.

43768 **EXAMPLES**43769 **Checking a Password Entry**

43770 The following example compares the information read from standard input to the value of the
 43771 name of the user entry. If the *strcmp()* function returns 0 (indicating a match), a further check
 43772 will be made to see if the user entered the proper old password. The *crypt()* function shall
 43773 encrypt the old password entered by the user, using the value of the encrypted password in the
 43774 **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the
 43775 structure, the entered password *oldpasswd* is the correct user's password. Finally, the program
 43776 encrypts the new password so that it can store the information in the **passwd** structure.

```

43777 #include <string.h>
43778 #include <unistd.h>
43779 #include <stdio.h>
43780 ...
43781 int valid_change;
43782 struct passwd *p;
43783 char user[100];
43784 char oldpasswd[100];
43785 char newpasswd[100];
43786 char savepasswd[100];
43787 ...
43788 if (strcmp(p->pw_name, user) == 0) {
43789     if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
43790         strcpy(savepasswd, crypt(newpasswd, user));
43791         p->pw_passwd = savepasswd;
43792         valid_change = 1;
43793     }
43794     else {

```

```
43795         fprintf(stderr, "Old password is not valid\n");
43796     }
43797 }
43798 ...
```

43799 **APPLICATION USAGE**

43800 None.

43801 **RATIONALE**

43802 None.

43803 **FUTURE DIRECTIONS**

43804 None.

43805 **SEE ALSO**

43806 *strncmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>

43807 **CHANGE HISTORY**

43808 First released in Issue 1. Derived from Issue 1 of the SVID.

43809 **Issue 6**

43810 Extensions beyond the ISO C standard are marked.

43811 **NAME**

43812 strcoll — string comparison using collating information

43813 **SYNOPSIS**

43814 #include <string.h>

43815 int strcoll(const char *s1, const char *s2);

43816 **DESCRIPTION**

43817 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43818 conflict between the requirements described here and the ISO C standard is unintentional. This
 43819 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43820 The *strcoll()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*,
 43821 both interpreted as appropriate to the *LC_COLLATE* category of the current locale.

43822 CX The *strcoll()* function shall not change the setting of *errno* if successful.

43823 Since no return value is reserved to indicate an error, an application wishing to check for error
 43824 situations should set *errno* to 0, then call *strcoll()*, then check *errno*.

43825 **RETURN VALUE**

43826 Upon successful completion, *strcoll()* shall return an integer greater than, equal to, or less than 0,
 43827 according to whether the string pointed to by *s1* is greater than, equal to, or less than the string
 43828 CX pointed to by *s2* when both are interpreted as appropriate to the current locale. On error,
 43829 *strcoll()* may set *errno*, but no return value is reserved to indicate an error.

43830 **ERRORS**43831 The *strcoll()* function may fail if:

43832 CX	[EINVAL]	The <i>s1</i> or <i>s2</i> arguments contain characters outside the domain of the collating
43833		sequence.

43834 **EXAMPLES**43835 **Comparing Nodes**

43836 The following example uses an application-defined function, *node_compare()*, to compare two
 43837 nodes based on an alphabetical ordering of the *string* field.

```

43838 #include <string.h>
43839 ...
43840 struct node { /* These are stored in the table. */
43841     char *string;
43842     int length;
43843 };
43844 ...
43845 int node_compare(const void *node1, const void *node2)
43846 {
43847     return strcoll(((const struct node *)node1)->string,
43848                   ((const struct node *)node2)->string);
43849 }
43850 ...
  
```

43851 **APPLICATION USAGE**43852 The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.

43853 **RATIONALE**

43854 None.

43855 **FUTURE DIRECTIONS**

43856 None.

43857 **SEE ALSO**43858 *strcmp()*, *strxfrm()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>43859 **CHANGE HISTORY**

43860 First released in Issue 3.

43861 **Issue 5**43862 The DESCRIPTION is updated to indicate that *errno* does not change if the function is
43863 successful.43864 **Issue 6**

43865 Extensions beyond the ISO C standard are marked.

43866 The following new requirements on POSIX implementations derive from alignment with the
43867 Single UNIX Specification:

- 43868
- The [EINVAL] optional error condition is added.

43869 An example is added.

43870 **NAME**

43871 strcpy — copy a string

43872 **SYNOPSIS**

43873 #include <string.h>

43874 char *strcpy(char *restrict s1, const char *restrict s2);

43875 **DESCRIPTION**

43876 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43877 conflict between the requirements described here and the ISO C standard is unintentional. This
 43878 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43879 The *strcpy()* function shall copy the string pointed to by *s2* (including the terminating null byte)
 43880 into the array pointed to by *s1*. If copying takes place between objects that overlap, the behavior
 43881 is undefined.

43882 **RETURN VALUE**43883 The *strcpy()* function shall return *s1*; no return value is reserved to indicate an error.43884 **ERRORS**

43885 No errors are defined.

43886 **EXAMPLES**43887 **Initializing a String**43888 The following example copies the string "-----" into the *permstring* variable.

```
43889       #include <string.h>
43890       ...
43891       static char permstring[11];
43892       ...
43893       strcpy(permstring, "-----");
43894       ...
```

43895 **Storing a Key and Data**

43896 The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the
 43897 key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there.
 43898 (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct**
 43899 **element** *.)

```
43900       #include <string.h>
43901       #include <stdlib.h>
43902       #include <stdio.h>
43903       ...
43904       /* Structure used to read data and store it. */
43905       struct element {
43906           char *key;
43907           char *data;
43908       };
43909       struct element *tbl, *curtbl;
43910       char *key, *data;
43911       int count;
43912       ...
43913       void dbfree(struct element *, int);
```



```
43914     ...
43915     if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
43916         perror("malloc"); dbfree(tbl, count); return NULL;
43917     }
43918     strcpy(curtbl->key, key);
43919     if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
43920         perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
43921     }
43922     strcpy(curtbl->data, data);
43923     ...
```

43924 **APPLICATION USAGE**

43925 Character movement is performed differently in different implementations. Thus, overlapping
43926 moves may yield surprises.

43927 This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3
43928 applications. Reliable error detection by this function was never guaranteed.

43929 **RATIONALE**

43930 None.

43931 **FUTURE DIRECTIONS**

43932 None.

43933 **SEE ALSO**

43934 *strncpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>

43935 **CHANGE HISTORY**

43936 First released in Issue 1. Derived from Issue 1 of the SVID.

43937 **Issue 6**

43938 The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43939 **NAME**

43940 strcspn — get the length of a complementary substring

43941 **SYNOPSIS**

43942 #include <string.h>

43943 size_t strcspn(const char *s1, const char *s2);

43944 **DESCRIPTION**

43945 cx The functionality described on this reference page is aligned with the ISO C standard. Any
43946 conflict between the requirements described here and the ISO C standard is unintentional. This
43947 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43948 The *strcspn()* function shall compute the length (in bytes) of the maximum initial segment of the
43949 string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.

43950 **RETURN VALUE**

43951 The *strcspn()* function shall return the length of the computed segment of the string pointed to
43952 by *s1*; no return value is reserved to indicate an error.

43953 **ERRORS**

43954 No errors are defined.

43955 **EXAMPLES**

43956 None.

43957 **APPLICATION USAGE**

43958 None.

43959 **RATIONALE**

43960 None.

43961 **FUTURE DIRECTIONS**

43962 None.

43963 **SEE ALSO**43964 *strspn()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>43965 **CHANGE HISTORY**

43966 First released in Issue 1. Derived from Issue 1 of the SVID.

43967 **Issue 5**

43968 The RETURN VALUE section is updated to indicate that *strcspn()* returns the length of *s1*, and
43969 not *s1* itself as was previously stated.

43970 **Issue 6**

43971 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is
43972 updated to indicate that the computed segment length is returned, not the *s1* length.

43973 **NAME**

43974 strdup — duplicate a string

43975 **SYNOPSIS**

43976 XSI #include <string.h>

43977 char *strdup(const char *s1);

43978

43979 **DESCRIPTION**

43980 The *strdup()* function shall return a pointer to a new string, which is a duplicate of the string
43981 pointed to by *s1*. The returned pointer can be passed to *free()*. A null pointer is returned if the
43982 new string cannot be created.

43983 **RETURN VALUE**

43984 The *strdup()* function shall return a pointer to a new string on success. Otherwise, it shall return
43985 a null pointer and set *errno* to indicate the error.

43986 **ERRORS**43987 The *strdup()* function may fail if:

43988 [ENOMEM] Storage space available is insufficient.

43989 **EXAMPLES**

43990 None.

43991 **APPLICATION USAGE**

43992 None.

43993 **RATIONALE**

43994 None.

43995 **FUTURE DIRECTIONS**

43996 None.

43997 **SEE ALSO**43998 *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>43999 **CHANGE HISTORY**

44000 First released in Issue 4, Version 2.

44001 **Issue 5**

44002 Moved from X/OPEN UNIX extension to BASE.

44003 **NAME**

44004 strerror, strerror_r — get error message string

44005 **SYNOPSIS**

44006 #include <string.h>

44007 char *strerror(int *errnum*);44008 TSF int strerror_r(int *errnum*, char **strerrbuf*, size_t *buflen*);

44009

44010 **DESCRIPTION**

44011 CX For *strerror()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44014 The *strerror()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return a pointer to it. Typically, the values for *errnum* come from *errno*, but *strerror()* shall map any value of type **int** to a message.

44017 The string pointed to shall not be modified by the application, but may be overwritten by a subsequent call to *strerror()* or *perror()*.

44019 CX The contents of the error message strings returned by *strerror()* should be determined by the setting of the *LC_MESSAGES* category in the current locale.

44021 The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *strerror()*.

44023 CX The *strerror()* function shall not change the setting of *errno* if successful.

44024 Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strerror()*, then check *errno*.

44026 The *strerror()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

44028 TSF The *strerror_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

44030

44031 **RETURN VALUE**

44032 Upon successful completion, *strerror()* shall return a pointer to the generated message string. On error *errno* may be set, but no return value is reserved to indicate an error.

44034 TSF Upon successful completion, *strerror_r()* shall return 0. Otherwise, an error number shall be returned to indicate the error.

44036 **ERRORS**

44037 These functions may fail if:

44038 [EINVAL] The value of *errnum* is not a valid error number.

44039 The *strerror_r()* function may fail if:

44040 TSF [ERANGE] Insufficient storage was supplied via *strerrbuf* and *buflen* to contain the generated message string.

44041

44042 **EXAMPLES**

44043 None.

44044 **APPLICATION USAGE**

44045 None.

44046 **RATIONALE**

44047 None.

44048 **FUTURE DIRECTIONS**

44049 None.

44050 **SEE ALSO**44051 *perror()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44052 **CHANGE HISTORY**

44053 First released in Issue 3.

44054 **Issue 5**44055 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

44056 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

44057 **Issue 6**

44058 Extensions beyond the ISO C standard are marked.

44059 The following new requirements on POSIX implementations derive from alignment with the
44060 Single UNIX Specification:

- 44061 • In the RETURN VALUE section, the fact that *errno* may be set is added.
- 44062 • The [EINVAL] optional error condition is added.

44063 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44064 The *strerror_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.44065 The *strerror_r()* function is marked as part of the Thread-Safe Functions option.

44066 **NAME**

44067 strfmon — convert monetary value to a string

44068 **SYNOPSIS**44069 xSI `#include <monetary.h>`

```
44070 ssize_t strfmon(char *restrict s, size_t maxsize,
44071               const char *restrict format, ...);
44072
```

44073 **DESCRIPTION**

44074 The *strfmon()* function shall place characters into the array pointed to by *s* as controlled by the
 44075 string pointed to by *format*. No more than *maxsize* bytes are placed into the array.

44076 The format is a character string, beginning and ending in its initial state, if any, that contains two
 44077 types of objects: *plain characters*, which are simply copied to the output stream, and *conversion*
 44078 *specifications*, each of which shall result in the fetching of zero or more arguments which are
 44079 converted and formatted. The results are undefined if there are insufficient arguments for the
 44080 format. If the format is exhausted while arguments remain, the excess arguments are simply
 44081 ignored.

44082 The application shall ensure that a conversion specification consists of the following sequence:

- 44083 • A '%' character
- 44084 • Optional flags
- 44085 • Optional field width
- 44086 • Optional left precision
- 44087 • Optional right precision
- 44088 • A required conversion specifier character that determines the conversion to be performed

44089 **Flags**

44090 One or more of the following optional flags can be specified to control the conversion:

- 44091 *=f* An '=' followed by a single character *f* which is used as the numeric fill character. In
 44092 order to work with precision or width counts, the fill character shall be a single byte
 44093 character; if not, the behavior is undefined. The default numeric fill character is the
 44094 <space>. This flag does not affect field width filling which always uses the <space>.
 44095 This flag is ignored unless a left precision (see below) is specified.
- 44096 ^ Do not format the currency amount with grouping characters. The default is to insert
 44097 the grouping characters if defined for the current locale.
- 44098 + or (Specify the style of representing positive and negative currency amounts. Only one of
 44099 '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '-'
 44100 are used (for example, in the U.S., the empty string if positive and '-' if negative). If
 44101 '(' is specified, negative amounts are enclosed within parentheses. If neither flag is
 44102 specified, the '+' style is used.
- 44103 ! Suppress the currency symbol from the output conversion.
- 44104 - Specify the alignment. If this flag is present the result of the conversion is left-justified
 44105 (padded to the right) rather than right-justified. This flag shall be ignored unless a field
 44106 width (see below) is specified.

44107 **Field Width**

44108 **w** A decimal digit string *w* specifying a minimum field width in bytes in which the result
 44109 of the conversion is right-justified (or left-justified if the flag ‘-’ is specified). The
 44110 default is 0.

44111 **Left Precision**

44112 **#n** A ‘#’ followed by a decimal digit string *n* specifying a maximum number of digits
 44113 expected to be formatted to the left of the radix character. This option can be used to
 44114 keep the formatted output from multiple calls to the *strfmon()* function aligned in the
 44115 same columns. It can also be used to fill unused positions with a special character as in
 44116 "\$***123.45". This option causes an amount to be formatted as if it has the number
 44117 of digits specified by *n*. If more than *n* digit positions are required, this conversion
 44118 specification is ignored. Digit positions in excess of those actually required are filled
 44119 with the numeric fill character (see the *=f* flag above).

44120 If grouping has not been suppressed with the ‘^’ flag, and it is defined for the current
 44121 locale, grouping separators are inserted before the fill characters (if any) are added.
 44122 Grouping separators are not applied to fill characters even if the fill character is a digit.

44123 To ensure alignment, any characters appearing before or after the number in the
 44124 formatted output such as currency or sign symbols are padded as necessary with
 44125 <space>s to make their positive and negative formats an equal length.

44126 **Right Precision**

44127 **.p** A period followed by a decimal digit string *p* specifying the number of digits after the
 44128 radix character. If the value of the right precision *p* is 0, no radix character appears. If a
 44129 right precision is not included, a default specified by the current locale is used. The
 44130 amount being formatted is rounded to the specified number of digits prior to
 44131 formatting.

44132 **Conversion Specifier Characters**

44133 The conversion specifier characters and their meanings are:

44134 **i** The **double** argument is formatted according to the locale’s international currency
 44135 format (for example, in the U.S.: USD 1,234.56). If the argument is $\pm\text{Inf}$ or NaN, the
 44136 result of the conversion is unspecified.

44137 **n** The **double** argument is formatted according to the locale’s national currency format
 44138 (for example, in the U.S.: \$1,234.56). If the argument is $\pm\text{Inf}$ or NaN, the result of the
 44139 conversion is unspecified.

44140 **%** Convert to a ‘%’; no argument is converted. The entire conversion specification shall
 44141 be %%.

44142 **Locale Information**

44143 The *LC_MONETARY* category of the program’s locale affects the behavior of this function
 44144 including the monetary radix character (which may be different from the numeric radix
 44145 character affected by the *LC_NUMERIC* category), the grouping separator, the currency
 44146 symbols, and formats. The international currency symbol should be conformant with the
 44147 ISO 4217:2001 standard.

44148 If the value of *maxsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

44149 **RETURN VALUE**

44150 If the total number of resulting bytes including the terminating null byte is not more than
 44151 *maxsize*, *strfmon()* shall return the number of bytes placed into the array pointed to by *s*, not
 44152 including the terminating null byte. Otherwise, -1 shall be returned, the contents of the array are
 44153 unspecified, and *errno* shall be set to indicate the error.

44154 **ERRORS**

44155 The *strfmon()* function shall fail if:

44156 [E2BIG] Conversion stopped due to lack of space in the buffer.

44157 **EXAMPLES**

44158 Given a locale for the U.S. and the values 123.45, -123.45, and 3456.781, the following output
 44159 might be produced. Square brackets (" [] ") are used in this example to delimit the output.

44160	%n	[\$123.45]	Default formatting
44161		[-\$123.45]	
44162		[\$3,456.78]	
44163	%11n	[\$123.45]	Right align within an 11-character field
44164		[-\$123.45]	
44165		[\$3,456.78]	
44166	%#5n	[\$ 123.45]	Aligned columns for values up to 99999
44167		[-\$ 123.45]	
44168		[\$ 3,456.78]	
44169	%=*#5n	[\$***123.45]	Specify a fill character
44170		[-\$***123.45]	
44171		[\$*3,456.78]	
44172	%=0#5n	[\$000123.45]	Fill characters do not use grouping
44173		[-\$000123.45]	even if the fill character is a digit
44174		[\$03,456.78]	
44175	%^#5n	[\$ 123.45]	Disable the grouping separator
44176		[-\$ 123.45]	
44177		[\$ 3456.78]	
44178	%^#5.0n	[\$ 123]	Round off to whole units
44179		[-\$ 123]	
44180		[\$ 3457]	
44181	%^#5.4n	[\$ 123.4500]	Increase the precision
44182		[-\$ 123.4500]	
44183		[\$ 3456.7810]	
44184	%(#5n	[\$ 123.45]	Use an alternative pos/neg style
44185		[(\$ 123.45)]	
44186		[\$ 3,456.78]	
44187	%!(#5n	[123.45]	Disable the currency symbol
44188		[(123.45)]	
44189		[3,456.78]	
44190	%-14#5.4n	[\$ 123.4500]	Left-justify the output
44191		[-\$ 123.4500]	
44192		[\$ 3,456.7810]	


```

44193      %14#5.4n    [  $    123.4500]    Corresponding right-justified output
44194                        [  -$    123.4500]
44195                        [  $ 3,456.7810]

```

44196 See also the EXAMPLES section in *fprintf()*.

44197 **APPLICATION USAGE**

44198 None.

44199 **RATIONALE**

44200 None.

44201 **FUTURE DIRECTIONS**

44202 Lowercase conversion characters are reserved for future standards use and uppercase for
 44203 implementation-defined use.

44204 **SEE ALSO**

44205 *fprintf()*, *localeconv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**monetary.h**>

44206 **CHANGE HISTORY**

44207 First released in Issue 4.

44208 **Issue 5**

44209 Moved from ENHANCED I18N to BASE.

44210 The [ENOSYS] error is removed.

44211 A sentence is added to the DESCRIPTION warning about values of *maxsize* that are greater than
 44212 {SSIZE_MAX}.

44213 **Issue 6**

44214 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44215 The **restrict** keyword is added to the *strfmon()* prototype for alignment with the
 44216 ISO/IEC 9899:1999 standard.

44217 The EXAMPLES section is reworked, clarifying the output format.

44218 **NAME**

44219 strftime — convert date and time to a string

44220 **SYNOPSIS**

44221 #include <time.h>

```
44222           size_t strftime(char *restrict s, size_t maxsize,
44223                           const char *restrict format, const struct tm *restrict timeptr);
```

44224 **DESCRIPTION**

44225 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 44226 conflict between the requirements described here and the ISO C standard is unintentional. This
 44227 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44228 The *strftime()* function shall place bytes into the array pointed to by *s* as controlled by the string
 44229 pointed to by *format*. The format is a character string, beginning and ending in its initial shift
 44230 state, if any. The *format* string consists of zero or more conversion specifications and ordinary
 44231 characters. A conversion specification consists of a '%' character, possibly followed by an E or O
 44232 modifier, and a terminating conversion specifier character that determines the conversion
 44233 specification's behavior. All ordinary characters (including the terminating null byte) are copied
 44234 unchanged into the array. If copying takes place between objects that overlap, the behavior is
 44235 undefined. No more than *maxsize* bytes are placed into the array. Each conversion specifier is
 44236 replaced by appropriate characters as described in the following list. The appropriate characters
 44237 are determined using the *LC_TIME* category of the current locale and by the values of zero or
 44238 more members of the broken-down time structure pointed to by *timeptr*, as specified in brackets
 44239 in the description. If any of the specified values are outside the normal range, the characters
 44240 stored are unspecified.

44241 CX Local timezone information is used as though *strftime()* called *tzset()*.

44242 The following conversion specifications are supported:

44243	%a	Replaced by the locale's abbreviated weekday name. [<i>tm_wday</i>]
44244	%A	Replaced by the locale's full weekday name. [<i>tm_wday</i>]
44245	%b	Replaced by the locale's abbreviated month name. [<i>tm_mon</i>]
44246	%B	Replaced by the locale's full month name. [<i>tm_mon</i>]
44247	%c	Replaced by the locale's appropriate date and time representation. (See the Base Definitions volume of IEEE Std 1003.1-2001, < time.h >.)
44248		
44249	%C	Replaced by the year divided by 100 and truncated to an integer, as a decimal number [00,99]. [<i>tm_year</i>]
44250		
44251	%d	Replaced by the day of the month as a decimal number [01,31]. [<i>tm_mday</i>]
44252	%D	Equivalent to %m/%d/%Y. [<i>tm_mon</i> , <i>tm_mday</i> , <i>tm_year</i>]
44253	%e	Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space. [<i>tm_mday</i>]
44254		
44255	%F	Equivalent to %Y-%m-%d (the ISO 8601:2000 standard date format). [<i>tm_year</i> , <i>tm_mon</i> , <i>tm_mday</i>]
44256		
44257	%g	Replaced by the last 2 digits of the week-based year (see below) as a decimal number [00,99]. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
44258		
44259	%G	Replaced by the week-based year (see below) as a decimal number (for example, 1977). [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
44260		

44261	%h	Equivalent to %b. [<i>tm_mon</i>]
44262	%H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [<i>tm_hour</i>]
44263	%I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [<i>tm_hour</i>]
44264	%j	Replaced by the day of the year as a decimal number [001,366]. [<i>tm_yday</i>]
44265	%m	Replaced by the month as a decimal number [01,12]. [<i>tm_mon</i>]
44266	%M	Replaced by the minute as a decimal number [00,59]. [<i>tm_min</i>]
44267	%n	Replaced by a <newline>.
44268	%p	Replaced by the locale's equivalent of either a.m. or p.m. [<i>tm_hour</i>]
44269 CX	%r	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be
44270		equivalent to %I:%M:%S %p. [<i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i>]
44271	%R	Replaced by the time in 24-hour notation (%H:%M). [<i>tm_hour</i> , <i>tm_min</i>]
44272	%S	Replaced by the second as a decimal number [00,60]. [<i>tm_sec</i>]
44273	%t	Replaced by a <tab>.
44274	%T	Replaced by the time (%H:%M:%S). [<i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i>]
44275	%u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday.
44276		[<i>tm_wday</i>]
44277	%U	Replaced by the week number of the year as a decimal number [00,53]. The first
44278		Sunday of January is the first day of week 1; days in the new year before this are in
44279		week 0. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
44280	%V	Replaced by the week number of the year (Monday as the first day of the week) as a
44281		decimal number [01,53]. If the week containing 1 January has four or more days in the
44282		new year, then it is considered week 1. Otherwise, it is the last week of the previous
44283		year, and the next week is week 1. Both January 4th and the first Thursday of January
44284		are always in week 1. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
44285	%w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.
44286		[<i>tm_wday</i>]
44287	%W	Replaced by the week number of the year as a decimal number [00,53]. The first
44288		Monday of January is the first day of week 1; days in the new year before this are in
44289		week 0. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
44290	%x	Replaced by the locale's appropriate date representation. (See the Base Definitions
44291		volume of IEEE Std 1003.1-2001, <time.h>.)
44292	%X	Replaced by the locale's appropriate time representation. (See the Base Definitions
44293		volume of IEEE Std 1003.1-2001, <time.h>.)
44294	%y	Replaced by the last two digits of the year as a decimal number [00,99]. [<i>tm_year</i>]
44295	%Y	Replaced by the year as a decimal number (for example, 1997). [<i>tm_year</i>]
44296	%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or
44297		-hhmm), or by no characters if no timezone is determinable. For example, "-0430"
44298 CX		means 4 hours 30 minutes behind UTC (west of Greenwich). If <i>tm_isdst</i> is zero, the
44299		standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight savings time
44300		offset is used. If <i>tm_isdst</i> is negative, no characters are returned. [<i>tm_isdst</i>]

44301 %Z Replaced by the timezone name or abbreviation, or by no bytes if no timezone
44302 information exists. [*tm_isdst*]

44303 %% Replaced by %.

44304 If a conversion specification does not correspond to any of the above, the behavior is undefined.

44305 CX If a **struct tm** broken-down time structure is created by *localtime()* or *localtime_r()*, or modified
44306 by *mktime()*, and the value of *TZ* is subsequently modified, the results of the %Z and %z *strftime()*
44307 conversion specifiers are undefined, when *strftime()* is called with such a broken-down time
44308 structure.

44309 If a **struct tm** broken-down time structure is created or modified by *gmtime()* or *gmtime_r()*, it is
44310 unspecified whether the result of the %Z and %z conversion specifiers shall refer to UTC or the
44311 current local timezone, when *strftime()* is called with such a broken-down time structure.

44312 **Modified Conversion Specifiers**

44313 Some conversion specifiers can be modified by the E or O modifier characters to indicate that an
44314 alternative format or specification should be used rather than the one normally used by the
44315 unmodified conversion specifier. If the alternative format or specification does not exist for the
44316 current locale (see ERA in the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5,
44317 LC_TIME), the behavior shall be as if the unmodified conversion specification were used.

44318 %Ec Replaced by the locale's alternative appropriate date and time representation.

44319 %EC Replaced by the name of the base year (period) in the locale's alternative
44320 representation.

44321 %Ex Replaced by the locale's alternative date representation.

44322 %EX Replaced by the locale's alternative time representation.

44323 %Ey Replaced by the offset from %EC (year only) in the locale's alternative representation.

44324 %EY Replaced by the full alternative year representation.

44325 %Od Replaced by the day of the month, using the locale's alternative numeric symbols, filled
44326 as needed with leading zeros if there is any alternative symbol for zero; otherwise, with
44327 leading spaces.

44328 %Oe Replaced by the day of the month, using the locale's alternative numeric symbols, filled
44329 as needed with leading spaces.

44330 %OH Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.

44331 %OI Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.

44332 %Om Replaced by the month using the locale's alternative numeric symbols.

44333 %OM Replaced by the minutes using the locale's alternative numeric symbols.

44334 %OS Replaced by the seconds using the locale's alternative numeric symbols.

44335 %Ou Replaced by the weekday as a number in the locale's alternative representation
44336 (Monday=1).

44337 %OU Replaced by the week number of the year (Sunday as the first day of the week, rules
44338 corresponding to %U) using the locale's alternative numeric symbols.

44339 %OV Replaced by the week number of the year (Monday as the first day of the week, rules
44340 corresponding to %V) using the locale's alternative numeric symbols.

44341 %Ow Replaced by the number of the weekday (Sunday=0) using the locale's alternative
 44342 numeric symbols.

44343 %OW Replaced by the week number of the year (Monday as the first day of the week) using
 44344 the locale's alternative numeric symbols.

44345 %Oy Replaced by the year (offset from %C) using the locale's alternative numeric symbols.

44346 %g, %G, and %V give values according to the ISO 8601:2000 standard week-based year. In this
 44347 system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th,
 44348 which is also the week that includes the first Thursday of the year, and is also the first week that
 44349 contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the
 44350 preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January
 44351 1999, %G is replaced by 1998 and %V is replaced by 53. If December 29th, 30th, or 31st is a
 44352 Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday
 44353 30th December 1997, %G is replaced by 1998 and %V is replaced by 01.

44354 If a conversion specifier is not one of the above, the behavior is undefined.

44355 RETURN VALUE

44356 If the total number of resulting bytes including the terminating null byte is not more than
 44357 *maxsize*, *strftime()* shall return the number of bytes placed into the array pointed to by *s*, not
 44358 including the terminating null byte. Otherwise, 0 shall be returned and the contents of the array
 44359 are unspecified.

44360 ERRORS

44361 No errors are defined.

44362 EXAMPLES

44363 **Getting a Localized Date String**

44364 The following example first sets the locale to the user's default. The locale information will be
 44365 used in the *nl_langinfo()* and *strftime()* functions. The *nl_langinfo()* function returns the localized
 44366 date string which specifies how the date is laid out. The *strftime()* function takes this information
 44367 and, using the **tm** structure for values, places the date and time information into *datestring*.

```
44368        #include <time.h>
44369        #include <locale.h>
44370        #include <langinfo.h>
44371        ...
44372        struct tm *tm;
44373        char datestring[256];
44374        ...
44375        setlocale (LC_ALL, "");
44376        ...
44377        strftime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
44378        ...
```

44379 APPLICATION USAGE

44380 The range of values for %S is [00,60] rather than [00,59] to allow for the occasional leap second.

44381 Some of the conversion specifications are duplicates of others. They are included for
 44382 compatibility with *nl_cxtime()* and *nl_ascxtime()*, which were published in Issue 2.

44383 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

44384 In the C locale, the E and O modifiers are ignored and the replacement strings for the following
 44385 specifiers are:

44386	%a	The first three characters of %A.
44387	%A	One of Sunday, Monday, ..., Saturday.
44388	%b	The first three characters of %B.
44389	%B	One of January, February, ..., December.
44390	%c	Equivalent to %a %b %e %T %Y.
44391	%p	One of AM or PM.
44392	%r	Equivalent to %I:%M:%S %p.
44393	%x	Equivalent to %m/%d/%Y.
44394	%X	Equivalent to %T.
44395	%Z	Implementation-defined.
44396	RATIONALE	
44397	None.	
44398	FUTURE DIRECTIONS	
44399	None.	
44400	SEE ALSO	
44401	<i>asctime()</i> , <i>clock()</i> , <i>ctime()</i> , <i>difftime()</i> , <i>getdate()</i> , <i>gmtime()</i> , <i>localtime()</i> , <i>mktime()</i> , <i>strptime()</i> , <i>time()</i> ,	
44402	<i>tzset()</i> , <i>utime()</i> , Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5, LC_TIME,	
44403	< time.h >	
44404	CHANGE HISTORY	
44405	First released in Issue 3.	
44406	Issue 5	
44407	The description of %OV is changed to be consistent with %V and defines Monday as the first day	
44408	of the week.	
44409	The description of %Oy is clarified.	
44410	Issue 6	
44411	Extensions beyond the ISO C standard are marked.	
44412	The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from	
44413	“Otherwise, it is week 53 of the previous year, and the next week is week 1” to “Otherwise, it is	
44414	the last week of the previous year, and the next week is week 1”.	
44415	The following new requirements on POSIX implementations derive from alignment with the	
44416	Single UNIX Specification:	
44417	• The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.	
44418	• The modified conversion specifiers are added for consistency with the ISO POSIX-2 standard	
44419	date utility.	
44420	The following changes are made for alignment with the ISO/IEC 9899:1999 standard:	
44421	• The <i>strptime()</i> prototype is updated.	
44422	• The DESCRIPTION is extensively revised.	
44423	• The %z conversion specifier is added.	
44424	A new example is added.	

44425

IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/60 is applied.

44426 NAME

44427 strlen — get string length

44428 SYNOPSIS

44429 #include <string.h>

44430 size_t strlen(const char *s);

44431 DESCRIPTION

44432 cx The functionality described on this reference page is aligned with the ISO C standard. Any
44433 conflict between the requirements described here and the ISO C standard is unintentional. This
44434 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44435 The *strlen()* function shall compute the number of bytes in the string to which *s* points, not
44436 including the terminating null byte.

44437 RETURN VALUE

44438 The *strlen()* function shall return the length of *s*; no return value shall be reserved to indicate an
44439 error.

44440 ERRORS

44441 No errors are defined.

44442 EXAMPLES

44443 Getting String Lengths

44444 The following example sets the maximum length of *key* and *data* by using *strlen()* to get the
44445 lengths of those strings.

```
44446       #include <string.h>
44447       ...
44448       struct element {
44449           char *key;
44450           char *data;
44451       };
44452       ...
44453       char *key, *data;
44454       int len;

44455       *keylength = *datalength = 0;
44456       ...
44457       if ((len = strlen(key)) > *keylength)
44458           *keylength = len;
44459       if ((len = strlen(data)) > *datalength)
44460           *datalength = len;
44461       ...
```

44462 APPLICATION USAGE

44463 None.

44464 RATIONALE

44465 None.

44466 FUTURE DIRECTIONS

44467 None.

44468 **SEE ALSO**

44469 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>

44470 **CHANGE HISTORY**

44471 First released in Issue 1. Derived from Issue 1 of the SVID.

44472 **Issue 5**

44473 The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not
44474 *s* itself as was previously stated.

44475 NAME

44476 strncasecmp — case-insensitive string comparison

44477 SYNOPSIS

44478 xSI #include <strings.h>

44479 int strncasecmp(const char *s1, const char *s2, size_t n);

44480

44481 DESCRIPTION

44482 Refer to *strcasecmp()*.

44483 **NAME**

44484 strncat — concatenate a string with part of another

44485 **SYNOPSIS**

44486 #include <string.h>

44487 char *strncat(char *restrict *s1*, const char *restrict *s2*, size_t *n*);44488 **DESCRIPTION**

44489 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
44490 conflict between the requirements described here and the ISO C standard is unintentional. This
44491 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44492 The *strncat()* function shall append not more than *n* bytes (a null byte and bytes that follow it
44493 are not appended) from the array pointed to by *s2* to the end of the string pointed to by *s1*. The
44494 initial byte of *s2* overwrites the null byte at the end of *s1*. A terminating null byte is always
44495 appended to the result. If copying takes place between objects that overlap, the behavior is
44496 undefined.

44497 **RETURN VALUE**44498 The *strncat()* function shall return *s1*; no return value shall be reserved to indicate an error.44499 **ERRORS**

44500 No errors are defined.

44501 **EXAMPLES**

44502 None.

44503 **APPLICATION USAGE**

44504 None.

44505 **RATIONALE**

44506 None.

44507 **FUTURE DIRECTIONS**

44508 None.

44509 **SEE ALSO**44510 *strcat()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44511 **CHANGE HISTORY**

44512 First released in Issue 1. Derived from Issue 1 of the SVID.

44513 **Issue 6**44514 The *strncat()* prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

44515 **NAME**

44516 strncmp — compare part of two strings

44517 **SYNOPSIS**

44518 #include <string.h>

44519 int strncmp(const char *s1, const char *s2, size_t n);

44520 **DESCRIPTION**

44521 cx The functionality described on this reference page is aligned with the ISO C standard. Any
44522 conflict between the requirements described here and the ISO C standard is unintentional. This
44523 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44524 The *strncmp()* function shall compare not more than *n* bytes (bytes that follow a null byte are not
44525 compared) from the array pointed to by *s1* to the array pointed to by *s2*.

44526 The sign of a non-zero return value is determined by the sign of the difference between the
44527 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings
44528 being compared.

44529 **RETURN VALUE**

44530 Upon successful completion, *strncmp()* shall return an integer greater than, equal to, or less than
44531 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the
44532 possibly null-terminated array pointed to by *s2* respectively.

44533 **ERRORS**

44534 No errors are defined.

44535 **EXAMPLES**

44536 None.

44537 **APPLICATION USAGE**

44538 None.

44539 **RATIONALE**

44540 None.

44541 **FUTURE DIRECTIONS**

44542 None.

44543 **SEE ALSO**44544 *strcmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44545 **CHANGE HISTORY**

44546 First released in Issue 1. Derived from Issue 1 of the SVID.

44547 **Issue 6**

44548 Extensions beyond the ISO C standard are marked.

44549 **NAME**

44550 strncpy — copy part of a string

44551 **SYNOPSIS**

44552 #include <string.h>

44553 char *strncpy(char *restrict *s1*, const char *restrict *s2*, size_t *n*);44554 **DESCRIPTION**

44555 cx The functionality described on this reference page is aligned with the ISO C standard. Any
44556 conflict between the requirements described here and the ISO C standard is unintentional. This
44557 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44558 The *strncpy()* function shall copy not more than *n* bytes (bytes that follow a null byte are not
44559 copied) from the array pointed to by *s2* to the array pointed to by *s1*. If copying takes place
44560 between objects that overlap, the behavior is undefined.

44561 If the array pointed to by *s2* is a string that is shorter than *n* bytes, null bytes shall be appended
44562 to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

44563 **RETURN VALUE**44564 The *strncpy()* function shall return *s1*; no return value is reserved to indicate an error.44565 **ERRORS**

44566 No errors are defined.

44567 **EXAMPLES**

44568 None.

44569 **APPLICATION USAGE**

44570 Character movement is performed differently in different implementations. Thus, overlapping
44571 moves may yield surprises.

44572 If there is no null byte in the first *n* bytes of the array pointed to by *s2*, the result is not null-
44573 terminated.

44574 **RATIONALE**

44575 None.

44576 **FUTURE DIRECTIONS**

44577 None.

44578 **SEE ALSO**44579 *strcpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44580 **CHANGE HISTORY**

44581 First released in Issue 1. Derived from Issue 1 of the SVID.

44582 **Issue 6**44583 The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

44584 **NAME**

44585 strpbrk — scan a string for a byte

44586 **SYNOPSIS**

44587 #include <string.h>

44588 char *strpbrk(const char *s1, const char *s2);

44589 **DESCRIPTION**

44590 cx The functionality described on this reference page is aligned with the ISO C standard. Any
44591 conflict between the requirements described here and the ISO C standard is unintentional. This
44592 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44593 The *strpbrk()* function shall locate the first occurrence in the string pointed to by *s1* of any byte
44594 from the string pointed to by *s2*.

44595 **RETURN VALUE**

44596 Upon successful completion, *strpbrk()* shall return a pointer to the byte or a null pointer if no
44597 byte from *s2* occurs in *s1*.

44598 **ERRORS**

44599 No errors are defined.

44600 **EXAMPLES**

44601 None.

44602 **APPLICATION USAGE**

44603 None.

44604 **RATIONALE**

44605 None.

44606 **FUTURE DIRECTIONS**

44607 None.

44608 **SEE ALSO**44609 *strchr()*, *strrchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>44610 **CHANGE HISTORY**

44611 First released in Issue 1. Derived from Issue 1 of the SVID.

44612 **NAME**

44613 strptime — date and time conversion

44614 **SYNOPSIS**44615 xSI `#include <time.h>`

```
44616 char *strptime(const char *restrict buf, const char *restrict format,
44617               struct tm *restrict tm);
44618
```

44619 **DESCRIPTION**

44620 The *strptime()* function shall convert the character string pointed to by *buf* to values which are
 44621 stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

44622 The *format* is composed of zero or more directives. Each directive is composed of one of the
 44623 following: one or more white-space characters (as specified by *isspace()*); an ordinary character
 44624 (neither '%' nor a white-space character); or a conversion specification. Each conversion
 44625 specification is composed of a '%' character followed by a conversion character which specifies
 44626 the replacement required. The application shall ensure that there is white-space or other non-
 44627 alphanumeric characters between any two conversion specifications. The following conversion
 44628 specifications are supported:

44629	%a	The day of the week, using the locale's weekday names; either the abbreviated or full name may be specified.
44630		
44631	%A	Equivalent to %a.
44632	%b	The month, using the locale's month names; either the abbreviated or full name may be specified.
44633		
44634	%B	Equivalent to %b.
44635	%c	Replaced by the locale's appropriate date and time representation.
44636	%C	The century number [00,99]; leading zeros are permitted but not required.
44637	%d	The day of the month [01,31]; leading zeros are permitted but not required.
44638	%D	The date as %m/%d/%y.
44639	%e	Equivalent to %d.
44640	%h	Equivalent to %b.
44641	%H	The hour (24-hour clock) [00,23]; leading zeros are permitted but not required.
44642	%I	The hour (12-hour clock) [01,12]; leading zeros are permitted but not required.
44643	%j	The day number of the year [001,366]; leading zeros are permitted but not required.
44644	%m	The month number [01,12]; leading zeros are permitted but not required.
44645	%M	The minute [00,59]; leading zeros are permitted but not required.
44646	%n	Any white space.
44647	%p	The locale's equivalent of a.m or p.m.
44648	%r	12-hour clock time using the AM/PM notation if t_fmt_ampm is not an empty string in the LC_TIME portion of the current locale; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
44649		
44650		
44651	%R	The time as %H:%M.

44652	%S	The seconds [00,60]; leading zeros are permitted but not required.
44653	%t	Any white space.
44654	%T	The time as %H:%M:%S.
44655	%U	The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.
44656		
44657	%w	The weekday as a decimal number [0,6], with 0 representing Sunday; leading zeros are permitted but not required.
44658		
44659	%W	The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.
44660		
44661	%x	The date, using the locale's date format.
44662	%X	The time, using the locale's time format.
44663	%y	The year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required.
44664		
44665		
44666		
44667	Note:	It is expected that in a future version of IEEE Std 1003.1-2001 the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
44668		
44669		
44670	%Y	The year, including the century (for example, 1988).
44671	%%	Replaced by %.

44672 **Modified Conversion Specifiers**

44673	Some conversion specifiers can be modified by the E and O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist in the current locale, the behavior shall be as if the unmodified conversion specification were used.	
44674		
44675		
44676		
44677	%Ec	The locale's alternative appropriate date and time representation.
44678	%EC	The name of the base year (period) in the locale's alternative representation.
44679	%Ex	The locale's alternative date representation.
44680	%EX	The locale's alternative time representation.
44681	%Ey	The offset from %EC (year only) in the locale's alternative representation.
44682	%EY	The full alternative year representation.
44683	%Od	The day of the month using the locale's alternative numeric symbols; leading zeros are permitted but not required.
44684		
44685	%Oe	Equivalent to %Od.
44686	%OH	The hour (24-hour clock) using the locale's alternative numeric symbols.
44687	%OI	The hour (12-hour clock) using the locale's alternative numeric symbols.
44688	%Om	The month using the locale's alternative numeric symbols.
44689	%OM	The minutes using the locale's alternative numeric symbols.

44690	%OS	The seconds using the locale's alternative numeric symbols.
44691	%OU	The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
44692		
44693	%Ow	The number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
44694	%OW	The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
44695		
44696	%Oy	The year (offset from %C) using the locale's alternative numeric symbols.
44697		A conversion specification composed of white-space characters is executed by scanning input up to the first character that is not white-space (which remains unscanned), or until no more characters can be scanned.
44698		
44699		
44700		A conversion specification that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the buffer differs from the one comprising the directive, the directive fails, and the differing and subsequent characters remain unscanned.
44701		
44702		
44703		
44704		A series of conversion specifications composed of %n, %t, white-space characters, or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.
44705		
44706		
44707		Any other conversion specification is executed by scanning characters until a character matching the next directive is scanned, or until no more characters can be scanned. These characters, except the one matching the next directive, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate tm structure members are set to values corresponding to the locale information. Case is ignored when matching items in <i>buf</i> such as month or weekday names. If no match is found, <i>strptime()</i> fails and no more characters are scanned.
44708		
44709		
44710		
44711		
44712		
44713		
44714	RETURN VALUE	
44715		Upon successful completion, <i>strptime()</i> shall return a pointer to the character following the last character parsed. Otherwise, a null pointer shall be returned.
44716		
44717	ERRORS	
44718		No errors are defined.
44719	EXAMPLES	
44720		None.
44721	APPLICATION USAGE	
44722		Several “equivalent to” formats and the special processing of white-space characters are provided in order to ease the use of identical <i>format</i> strings for <i>strftime()</i> and <i>strptime()</i> .
44723		
44724		Applications should use %Y (4-digit years) in preference to %y (2-digit years).
44725		It is unspecified whether multiple calls to <i>strptime()</i> using the same tm structure will update the current contents of the structure or overwrite all contents of the structure. Conforming applications should make a single call to <i>strptime()</i> with a format and all data needed to completely specify the date and time being converted.
44726		
44727		
44728		
44729	RATIONALE	
44730		None.

44731 **FUTURE DIRECTIONS**

44732 The *strptime()* function is expected to be mandatory in the next version of this volume of
44733 IEEE Std 1003.1-2001.

44734 **SEE ALSO**

44735 *scanf()*, *strptime()*, *time()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

44736 **CHANGE HISTORY**

44737 First released in Issue 4.

44738 **Issue 5**

44739 Moved from ENHANCED I18N to BASE.

44740 The [ENOSYS] error is removed.

44741 The exact meaning of the %y and %Oy specifiers is clarified in the DESCRIPTION.

44742 **Issue 6**

44743 The Open Group Corrigendum U033/5 is applied. The %r specifier description is reworded.

44744 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44745 The **restrict** keyword is added to the *strptime()* prototype for alignment with the
44746 ISO/IEC 9899:1999 standard.

44747 The Open Group Corrigendum U047/2 is applied.

44748 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
44749 specification” for consistency with *strptime()*.

44750 **NAME**

44751 strrchr — string scanning operation

44752 **SYNOPSIS**

44753 #include <string.h>

44754 char *strrchr(const char *s, int c);

44755 **DESCRIPTION**

44756 cx The functionality described on this reference page is aligned with the ISO C standard. Any
44757 conflict between the requirements described here and the ISO C standard is unintentional. This
44758 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44759 The *strrchr()* function shall locate the last occurrence of *c* (converted to a **char**) in the string
44760 pointed to by *s*. The terminating null byte is considered to be part of the string.

44761 **RETURN VALUE**

44762 Upon successful completion, *strrchr()* shall return a pointer to the byte or a null pointer if *c* does
44763 not occur in the string.

44764 **ERRORS**

44765 No errors are defined.

44766 **EXAMPLES**44767 **Finding the Base Name of a File**

44768 The following example uses *strrchr()* to get a pointer to the base name of a file. The *strrchr()*
44769 function searches backwards through the name of the file to find the last '/' character in *name*.
44770 This pointer (plus one) will point to the base name of the file.

```
44771       #include <string.h>
44772       ...
44773       const char *name;
44774       char *basename;
44775       ...
44776       basename = strrchr(name, '/') + 1;
44777       ...
```

44778 **APPLICATION USAGE**

44779 None.

44780 **RATIONALE**

44781 None.

44782 **FUTURE DIRECTIONS**

44783 None.

44784 **SEE ALSO**44785 *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44786 **CHANGE HISTORY**

44787 First released in Issue 1. Derived from Issue 1 of the SVID.

44788 **NAME**

44789 strspn — get length of a substring

44790 **SYNOPSIS**

44791 #include <string.h>

44792 size_t strspn(const char *s1, const char *s2);

44793 **DESCRIPTION**

44794 cx The functionality described on this reference page is aligned with the ISO C standard. Any
44795 conflict between the requirements described here and the ISO C standard is unintentional. This
44796 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44797 The *strspn()* function shall compute the length (in bytes) of the maximum initial segment of the
44798 string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

44799 **RETURN VALUE**

44800 The *strspn()* function shall return the length of *s1*; no return value is reserved to indicate an
44801 error.

44802 **ERRORS**

44803 No errors are defined.

44804 **EXAMPLES**

44805 None.

44806 **APPLICATION USAGE**

44807 None.

44808 **RATIONALE**

44809 None.

44810 **FUTURE DIRECTIONS**

44811 None.

44812 **SEE ALSO**44813 *strcspn()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44814 **CHANGE HISTORY**

44815 First released in Issue 1. Derived from Issue 1 of the SVID.

44816 **Issue 5**

44817 The RETURN VALUE section is updated to indicate that *strspn()* returns the length of *s*, and not
44818 *s* itself as was previously stated.

44819 **NAME**

44820 strstr — find a substring

44821 **SYNOPSIS**

44822 #include <string.h>

44823 char *strstr(const char *s1, const char *s2);

44824 **DESCRIPTION**

44825 cx The functionality described on this reference page is aligned with the ISO C standard. Any
44826 conflict between the requirements described here and the ISO C standard is unintentional. This
44827 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44828 The *strstr()* function shall locate the first occurrence in the string pointed to by *s1* of the
44829 sequence of bytes (excluding the terminating null byte) in the string pointed to by *s2*.

44830 **RETURN VALUE**

44831 Upon successful completion, *strstr()* shall return a pointer to the located string or a null pointer
44832 if the string is not found.

44833 If *s2* points to a string with zero length, the function shall return *s1*.

44834 **ERRORS**

44835 No errors are defined.

44836 **EXAMPLES**

44837 None.

44838 **APPLICATION USAGE**

44839 None.

44840 **RATIONALE**

44841 None.

44842 **FUTURE DIRECTIONS**

44843 None.

44844 **SEE ALSO**44845 *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44846 **CHANGE HISTORY**

44847 First released in Issue 3. Included for alignment with the ANSI C standard.

NAME

strtod, strtodf, strtold — convert a string to a double-precision number

SYNOPSIS

```
#include <stdlib.h>
```

```
double strtod(const char *restrict nptr, char **restrict endptr);
```

```
float strtodf(const char *restrict nptr, char **restrict endptr);
```

```
long double strtold(const char *restrict nptr, char **restrict endptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
3. A final string of one or more unrecognized characters, including the terminating null byte of the input string

Then they shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character, then an optional exponent part
- A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character, then an optional binary exponent part
- One of INF or INFINITY, ignoring case
- One of NAN or NAN(*n-char-sequence_{opt}*), ignoring case in the NAN part, where:

```
n-char-sequence:
    digit
    nondigit
    n-char-sequence digit
    n-char-sequence nondigit
```

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) shall be interpreted as a floating constant of the C language, except that the radix character shall be used in place of a period, and that if neither an exponent part nor a radix character appears in a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an

44892 infinity, if representable in the return type, else as if it were a floating constant that is too large
 44893 for the range of the return type. A character sequence NAN or NAN(*n-char-sequence_{opt}*) shall be
 44894 interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence
 44895 part that does not have the expected form; the meaning of the *n-char* sequences is
 44896 implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*,
 44897 provided that *endptr* is not a null pointer.

44898 If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value
 44899 resulting from the conversion is correctly rounded.

44900 CX The radix character is defined in the program's locale (category *LC_NUMERIC*). In the POSIX
 44901 locale, or in a locale where the radix character is not defined, the radix character shall default to a
 44902 period ('.').

44903 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
 44904 accepted.

44905 If the subject sequence is empty or does not have the expected form, no conversion shall be
 44906 performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not
 44907 a null pointer.

44908 CX The *strtod()* function shall not change the setting of *errno* if successful.

44909 Since 0 is returned on error and is also a valid return on success, an application wishing to check
 44910 for error situations should set *errno* to 0, then call *strtod()*, *strtof()*, or *strtold()*, then check *errno*.

44911 RETURN VALUE

44912 Upon successful completion, these functions shall return the converted value. If no conversion
 44913 could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

44914 If the correct value is outside the range of representable values, $\pm$ HUGE_VAL, $\pm$ HUGE_VALF, or |
 44915 $\pm$ HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to |
 44916 [ERANGE].

44917 If the correct value would cause an underflow, a value whose magnitude is no greater than the
 44918 smallest normalized positive number in the return type shall be returned and *errno* set to
 44919 [ERANGE].

44920 ERRORS

44921 These functions shall fail if:

44922 CX [ERANGE] The value to be returned would cause overflow or underflow.

44923 These functions may fail if:

44924 CX [EINVAL] No conversion could be performed.

44925 EXAMPLES

44926 None.

44927 APPLICATION USAGE

44928 If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the
 44929 result is not exactly representable, the result should be one of the two numbers in the
 44930 appropriate internal format that are adjacent to the hexadecimal floating source value, with the
 44931 extra stipulation that the error should have a correct sign for the current rounding direction.

44932 If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in <float.h>)
 44933 significant digits, the result should be correctly rounded. If the subject sequence *D* has the
 44934 decimal form and more than DECIMAL_DIG significant digits, consider the two bounding,
 44935 adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the

values of L , D , and U satisfy $L \leq D \leq U$. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding L and U according to the current rounding direction, with the extra stipulation that the error with respect to D should have a correct sign for the current rounding direction.

The changes to `strtod()` introduced by the ISO/IEC 9899:1999 standard can alter the behavior of well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier versions of the base documents. One such example would be:

```
int
what_kind_of_number (char *s)
{
    char *endp;
    double d;
    long l;

    d = strtod(s, &endp);
    if (s != endp && *endp == '\\0')
        printf("It's a float with value %g\\n", d);
    else
    {
        l = strtol(s, &endp, 0);
        if (s != endp && *endp == '\\0')
            printf("It's an integer with value %ld\\n", l);
        else
            return 1;
    }
    return 0;
}
```

If the function is called with:

```
what_kind_of_number ("0x10")
```

an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

```
It's an integer with value 16
```

With the ISO/IEC 9899:1999 standard, the result is:

```
It's a float with value 16
```

The change in behavior is due to the inclusion of floating-point numbers in hexadecimal notation without requiring that either a decimal point or the binary exponent be present.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

`isspace()`, `localeconv()`, `scanf()`, `setlocale()`, `strtol()`, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<float.h>**, **<stdlib.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

44979 **Issue 5**

44980 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

44981 **Issue 6**

44982 Extensions beyond the ISO C standard are marked.

44983 The following new requirements on POSIX implementations derive from alignment with the
44984 Single UNIX Specification:

- 44985 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
44986 added if no conversion could be performed.

44987 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 44988 • The *strtod()* function is updated.
- 44989 • The *strtof()* and *strtold()* functions are added.
- 44990 • The DESCRIPTION is extensively revised.

44991 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

44992 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second |
44993 paragraph in the RETURN VALUE section. This change clarifies the sign of the return value. |

44994 NAME

44995 strtoimax, strtoumax — convert string to integer type

44996 SYNOPSIS

44997 #include <inttypes.h>

44998 intmax_t strtoimax(const char *restrict nptr, char **restrict endptr,
44999 int base);

45000 uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
45001 int base);

45002 DESCRIPTION

45003 CX The functionality described on this reference page is aligned with the ISO C standard. Any
45004 conflict between the requirements described here and the ISO C standard is unintentional. This
45005 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45006 These functions shall be equivalent to the *strtol()*, *strtoll()*, *strtoul()*, and *strtoull()* functions,
45007 except that the initial portion of the string shall be converted to **intmax_t** and **uintmax_t**
45008 representation, respectively.

45009 RETURN VALUE

45010 These functions shall return the converted value, if any.

45011 If no conversion could be performed, zero shall be returned.

45012 If the correct value is outside the range of representable values, {INTMAX_MAX},
45013 {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign
45014 of the value, if any), and *errno* shall be set to [ERANGE].

45015 ERRORS

45016 These functions shall fail if:

45017 [ERANGE] The value to be returned is not representable.

45018 These functions may fail if:

45019 [EINVAL] The value of *base* is not supported.

45020 EXAMPLES

45021 None.

45022 APPLICATION USAGE

45023 None.

45024 RATIONALE

45025 None.

45026 FUTURE DIRECTIONS

45027 None.

45028 SEE ALSO

45029 *strtol()*, *strtoul()*, the Base Definitions volume of IEEE Std 1003.1-2001, <inttypes.h>

45030 CHANGE HISTORY

45031 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

45032 **NAME**45033 *strtok*, *strtok_r* — split string into tokens45034 **SYNOPSIS**

45035 #include <string.h>

45036 char **strtok*(char *restrict *s1*, const char *restrict *s2*);45037 TSF char **strtok_r*(char *restrict *s*, const char *restrict *sep*,45038 char **restrict *lasts*);

45039

45040 **DESCRIPTION**

45041 CX For *strtok()*: The functionality described on this reference page is aligned with the ISO C
 45042 standard. Any conflict between the requirements described here and the ISO C standard is
 45043 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45044 A sequence of calls to *strtok()* breaks the string pointed to by *s1* into a sequence of tokens, each
 45045 of which is delimited by a byte from the string pointed to by *s2*. The first call in the sequence has
 45046 *s1* as its first argument, and is followed by calls with a null pointer as their first argument. The
 45047 separator string pointed to by *s2* may be different from call to call.

45048 The first call in the sequence searches the string pointed to by *s1* for the first byte that is *not*
 45049 contained in the current separator string pointed to by *s2*. If no such byte is found, then there
 45050 are no tokens in the string pointed to by *s1* and *strtok()* shall return a null pointer. If such a byte
 45051 is found, it is the start of the first token.

45052 The *strtok()* function then searches from there for a byte that is contained in the current
 45053 separator string. If no such byte is found, the current token extends to the end of the string
 45054 pointed to by *s1*, and subsequent searches for a token shall return a null pointer. If such a byte is
 45055 found, it is overwritten by a null byte, which terminates the current token. The *strtok()* function
 45056 saves a pointer to the following byte, from which the next search for a token shall start.

45057 Each subsequent call, with a null pointer as the value of the first argument, starts searching from
 45058 the saved pointer and behaves as described above.

45059 The implementation shall behave as if no function defined in this volume of
 45060 IEEE Std 1003.1-2001 calls *strtok()*.

45061 CX The *strtok()* function need not be reentrant. A function that is not required to be reentrant is not
 45062 required to be thread-safe.

45063 TSF The *strtok_r()* function considers the null-terminated string *s* as a sequence of zero or more text
 45064 tokens separated by spans of one or more characters from the separator string *sep*. The
 45065 argument *lasts* points to a user-provided pointer which points to stored information necessary
 45066 for *strtok_r()* to continue scanning the same string.

45067 In the first call to *strtok_r()*, *s* points to a null-terminated string, *sep* to a null-terminated string of
 45068 separator characters, and the value pointed to by *lasts* is ignored. The *strtok_r()* function shall
 45069 return a pointer to the first character of the first token, write a null character into *s* immediately
 45070 following the returned token, and update the pointer to which *lasts* points.

45071 In subsequent calls, *s* is a NULL pointer and *lasts* shall be unchanged from the previous call so
 45072 that subsequent calls shall move through the string *s*, returning successive tokens until no
 45073 tokens remain. The separator string *sep* may be different from call to call. When no token
 45074 remains in *s*, a NULL pointer shall be returned.

45075 **RETURN VALUE**

45076 Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise,
 45077 if there is no token, *strtok()* shall return a null pointer.

45078 TSF The *strtok_r()* function shall return a pointer to the token found, or a NULL pointer when no
 45079 token is found.

45080 **ERRORS**

45081 No errors are defined.

45082 **EXAMPLES**45083 **Searching for Word Separators**

45084 The following example searches for tokens separated by <space>s.

```
45085 #include <string.h>
45086 ...
45087 char *token;
45088 char *line = "LINE TO BE SEPARATED";
45089 char *search = " ";

45090 /* Token will point to "LINE". */
45091 token = strtok(line, search);

45092 /* Token will point to "TO". */
45093 token = strtok(NULL, search);
```

45094 **Breaking a Line**

45095 The following example uses *strtok()* to break a line into two character strings separated by any
 45096 combination of <space>s, <tab>s, or <newline>s.

```
45097 #include <string.h>
45098 ...
45099 struct element {
45100     char *key;
45101     char *data;
45102 };
45103 ...
45104 char line[LINE_MAX];
45105 char *key, *data;
45106 ...
45107 key = strtok(line, " \n");
45108 data = strtok(NULL, " \n");
45109 ...
```

45110 **APPLICATION USAGE**

45111 The *strtok_r()* function is thread-safe and stores its state in a user-supplied buffer instead of
 45112 possibly using a static data area that may be overwritten by an unrelated call from another
 45113 thread.

45114 **RATIONALE**

45115 The *strtok()* function searches for a separator string within a larger string. It returns a pointer to
 45116 the last substring between separator strings. This function uses static storage to keep track of
 45117 the current string position between calls. The new function, *strtok_r()*, takes an additional
 45118 argument, *lasts*, to keep track of the current position in the string.

45119 **FUTURE DIRECTIONS**

45120 None.

45121 **SEE ALSO**45122 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>45123 **CHANGE HISTORY**

45124 First released in Issue 1. Derived from Issue 1 of the SVID.

45125 **Issue 5**45126 The *strtok_r()* function is included for alignment with the POSIX Threads Extension.45127 A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.45128 **Issue 6**

45129 Extensions beyond the ISO C standard are marked.

45130 The *strtok_r()* function is marked as part of the Thread-Safe Functions option.

45131 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

45132 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
45133 its avoidance of possibly using a static data area.45134 The **restrict** keyword is added to the *strtok()* and *strtok_r()* prototypes for alignment with the
45135 ISO / IEC 9899:1999 standard.

45136 **NAME**

45137 strtol, strtoll — convert a string to a long integer

45138 **SYNOPSIS**

45139 #include <stdlib.h>

45140 long strtol(const char *restrict *str*, char **restrict *endptr*, int *base*);

45141 long long strtoll(const char *restrict *str*, char **restrict *endptr*,

45142 int *base*)

45143 **DESCRIPTION**

45144 CX The functionality described on this reference page is aligned with the ISO C standard. Any
45145 conflict between the requirements described here and the ISO C standard is unintentional. This
45146 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45147 These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and
45148 **long long** representation, respectively. First, they decompose the input string into three parts:

- 45149 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 45150 2. A subject sequence interpreted as an integer represented in some radix determined by the
45151 value of *base*
- 45152 3. A final string of one or more unrecognized characters, including the terminating null byte
45153 of the input string.

45154 Then they shall attempt to convert the subject sequence to an integer, and return the result.

45155 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,
45156 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A
45157 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
45158 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to
45159 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
45160 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

45161 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
45162 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
45163 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the
45164 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the
45165 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and
45166 digits, following the sign if present.

45167 The subject sequence is defined as the longest initial subsequence of the input string, starting
45168 with the first non-white-space character that is of the expected form. The subject sequence shall
45169 contain no characters if the input string is empty or consists entirely of white-space characters,
45170 or if the first non-white-space character is other than a sign or a permissible letter or digit.

45171 If the subject sequence has the expected form and the value of *base* is 0, the sequence of
45172 characters starting with the first digit shall be interpreted as an integer constant. If the subject
45173 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the
45174 base for conversion, ascribing to each letter its value as given above. If the subject sequence
45175 begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to
45176 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null
45177 pointer.

45178 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
45179 accepted.

45180 If the subject sequence is empty or does not have the expected form, no conversion is performed;
 45181 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null
 45182 pointer.

45183 CX The *strtol()* function shall not change the setting of *errno* if successful.

45184 Since 0, {LONG_MIN} or {LLONG_MIN}, and {LONG_MAX} or {LLONG_MAX} are returned on
 45185 error and are also valid returns on success, an application wishing to check for error situations
 45186 should set *errno* to 0, then call *strtol()* or *strtoll()*, then check *errno*.

45187 RETURN VALUE

45188 Upon successful completion, these functions shall return the converted value, if any. If no
 45189 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

45190 If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
 45191 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
 45192 *errno* set to [ERANGE].

45193 ERRORS

45194 These functions shall fail if:

45195 [ERANGE] The value to be returned is not representable.

45196 These functions may fail if:

45197 CX [EINVAL] The value of *base* is not supported.

45198 EXAMPLES

45199 None.

45200 APPLICATION USAGE

45201 None.

45202 RATIONALE

45203 None.

45204 FUTURE DIRECTIONS

45205 None.

45206 SEE ALSO

45207 *isalpha()*, *scanf()*, *strtod()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

45208 CHANGE HISTORY

45209 First released in Issue 1. Derived from Issue 1 of the SVID.

45210 Issue 5

45211 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

45212 Issue 6

45213 Extensions beyond the ISO C standard are marked.

45214 The following new requirements on POSIX implementations derive from alignment with the
 45215 Single UNIX Specification:

- 45216 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
- 45217 added if no conversion could be performed.

45218 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 45219 • The *strtol()* prototype is updated.
- 45220 • The *strtoll()* function is added.

45221 **NAME**45222 **strtold** — convert a string to a double-precision number45223 **SYNOPSIS**45224 `#include <stdlib.h>`45225 `long double strtold(const char *restrict nptr, char **restrict endptr);`45226 **DESCRIPTION**45227 Refer to *strtod()*.

45228 **NAME**45229 `strtoll` — convert a string to a long integer45230 **SYNOPSIS**45231 `#include <stdlib.h>`45232 `long long strtoll(const char *restrict str, char **restrict endptr,`
45233 `int base);`45234 **DESCRIPTION**45235 Refer to *strtol()*.

45236 **NAME**

45237 strtol, strtoull — convert a string to an unsigned long

45238 **SYNOPSIS**

45239 #include <stdlib.h>

```

45240       unsigned long strtol(const char *restrict str,
45241                           char **restrict endptr, int base);
45242       unsigned long long strtoull(const char *restrict str,
45243                                   char **restrict endptr, int base);

```

45244 **DESCRIPTION**

45245 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 45246 conflict between the requirements described here and the ISO C standard is unintentional. This
 45247 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45248 These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned**
 45249 **long** and **unsigned long long** representation, respectively. First, they decompose the input
 45250 string into three parts:

- 45251 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 45252 2. A subject sequence interpreted as an integer represented in some radix determined by the
 45253 value of *base*
- 45254 3. A final string of one or more unrecognized characters, including the terminating null byte
 45255 of the input string

45256 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the
 45257 result.

45258 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,
 45259 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A
 45260 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
 45261 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to
 45262 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
 45263 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

45264 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
 45265 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
 45266 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the
 45267 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the
 45268 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and
 45269 digits, following the sign if present.

45270 The subject sequence is defined as the longest initial subsequence of the input string, starting
 45271 with the first non-white-space character that is of the expected form. The subject sequence shall
 45272 contain no characters if the input string is empty or consists entirely of white-space characters,
 45273 or if the first non-white-space character is other than a sign or a permissible letter or digit.

45274 If the subject sequence has the expected form and the value of *base* is 0, the sequence of
 45275 characters starting with the first digit shall be interpreted as an integer constant. If the subject
 45276 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the
 45277 base for conversion, ascribing to each letter its value as given above. If the subject sequence
 45278 begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to
 45279 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null
 45280 pointer.

45281 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
 45282 accepted.

45283 If the subject sequence is empty or does not have the expected form, no conversion shall be
 45284 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*
 45285 is not a null pointer.

45286 CX The *strtoul()* function shall not change the setting of *errno* if successful.

45287 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and are also valid returns
 45288 on success, an application wishing to check for error situations should set *errno* to 0, then call
 45289 *strtoul()* or *strtoull()*, then check *errno*.

45290 RETURN VALUE

45291 Upon successful completion, these functions shall return the converted value, if any. If no
 45292 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL]. If the
 45293 correct value is outside the range of representable values, {ULONG_MAX} or {ULLONG_MAX}
 45294 shall be returned and *errno* set to [ERANGE].

45295 ERRORS

45296 These functions shall fail if:

45297 CX [EINVAL] The value of *base* is not supported.

45298 [ERANGE] The value to be returned is not representable.

45299 These functions may fail if:

45300 CX [EINVAL] No conversion could be performed.

45301 EXAMPLES

45302 None.

45303 APPLICATION USAGE

45304 None.

45305 RATIONALE

45306 None.

45307 FUTURE DIRECTIONS

45308 None.

45309 SEE ALSO

45310 *isalpha()*, *scanf()*, *strtod()*, *strtol()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 45311 <stdlib.h>

45312 CHANGE HISTORY

45313 First released in Issue 4. Derived from the ANSI C standard.

45314 Issue 5

45315 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

45316 Issue 6

45317 Extensions beyond the ISO C standard are marked.

45318 The following new requirements on POSIX implementations derive from alignment with the
 45319 Single UNIX Specification:

- 45320 • The [EINVAL] error condition is added for when the value of *base* is not supported.

45321 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
 45322 added if no conversion could be performed.

- 45323 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 45324 • The *strtoul()* prototype is updated.
- 45325 • The *strtoull()* function is added.

45326 **NAME**45327 `strtoumax` — convert a string to an integer type45328 **SYNOPSIS**45329 `#include <inttypes.h>`45330 `uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,`
45331 `int base);`45332 **DESCRIPTION**45333 Refer to *strtoimax()*.

45334 **NAME**

45335 strxfrm — string transformation

45336 **SYNOPSIS**

45337 #include <string.h>

45338 size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);

45339 **DESCRIPTION**

45340 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 45341 conflict between the requirements described here and the ISO C standard is unintentional. This
 45342 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45343 The *strxfrm()* function shall transform the string pointed to by *s2* and place the resulting string
 45344 into the array pointed to by *s1*. The transformation is such that if *strcmp()* is applied to two
 45345 transformed strings, it shall return a value greater than, equal to, or less than 0, corresponding to
 45346 the result of *strcoll()* applied to the same two original strings. No more than *n* bytes are placed
 45347 into the resulting array pointed to by *s1*, including the terminating null byte. If *n* is 0, *s1* is
 45348 permitted to be a null pointer. If copying takes place between objects that overlap, the behavior
 45349 is undefined.

45350 CX The *strxfrm()* function shall not change the setting of *errno* if successful.

45351 Since no return value is reserved to indicate an error, an application wishing to check for error
 45352 situations should set *errno* to 0, then call *strxfrm()*, then check *errno*.

45353 **RETURN VALUE**

45354 Upon successful completion, *strxfrm()* shall return the length of the transformed string (not
 45355 including the terminating null byte). If the value returned is *n* or more, the contents of the array
 45356 pointed to by *s1* are unspecified.

45357 CX On error, *strxfrm()* may set *errno* but no return value is reserved to indicate an error.

45358 **ERRORS**

45359 The *strxfrm()* function may fail if:

45360 CX	[EINVAL]	The string pointed to by the <i>s2</i> argument contains characters outside the domain of the collating sequence.
45361		

45362 **EXAMPLES**

45363 None.

45364 **APPLICATION USAGE**

45365 The transformation function is such that two transformed strings can be ordered by *strcmp()* as
 45366 appropriate to collating sequence information in the program's locale (category *LC_COLLATE*).

45367 The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the
 45368 *s1* array prior to making the transformation.

45369 **RATIONALE**

45370 None.

45371 **FUTURE DIRECTIONS**

45372 None.

45373 **SEE ALSO**

45374 *strcmp()*, *strcoll()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>

45375 **CHANGE HISTORY**

45376 First released in Issue 3. Included for alignment with the ISO C standard.

45377 **Issue 5**

45378 The DESCRIPTION is updated to indicate that *errno* does not change if the function is
45379 successful.

45380 **Issue 6**

45381 Extensions beyond the ISO C standard are marked.

45382 The following new requirements on POSIX implementations derive from alignment with the
45383 Single UNIX Specification:

- 45384 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
45385 added if no conversion could be performed.

45386 The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

45387 **NAME**

45388 swab — swap bytes

45389 **SYNOPSIS**

45390 xSI #include <unistd.h>

45391 void swab(const void *restrict *src*, void *restrict *dest*,
45392 ssize_t *nbytes*);

45393

45394 **DESCRIPTION**

45395 The *swab()* function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to
45396 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab()*
45397 copies and exchanges *nbytes*–1 bytes and the disposition of the last byte is unspecified. If
45398 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is
45399 negative, *swab()* does nothing.

45400 **RETURN VALUE**

45401 None.

45402 **ERRORS**

45403 No errors are defined.

45404 **EXAMPLES**

45405 None.

45406 **APPLICATION USAGE**

45407 None.

45408 **RATIONALE**

45409 None.

45410 **FUTURE DIRECTIONS**

45411 None.

45412 **SEE ALSO**

45413 The Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

45414 **CHANGE HISTORY**

45415 First released in Issue 1. Derived from Issue 1 of the SVID.

45416 **Issue 6**

45417 The **restrict** keyword is added to the *swab()* prototype for alignment with the
45418 ISO/IEC 9899:1999 standard.

45419 **NAME**

45420 swapcontext — swap user context

45421 **SYNOPSIS**

45422 xSI #include <ucontext.h>

45423 int swapcontext(ucontext_t *restrict oucp,
45424 const ucontext_t *restrict ucp);

45425

45426 **DESCRIPTION**45427 Refer to *makecontext()*.

45428 NAME

45429 swprintf — print formatted wide-character output

45430 SYNOPSIS

45431 #include <stdio.h>

45432 #include <wchar.h>

45433 int swprintf(wchar_t *restrict *ws*, size_t *n*,

45434 const wchar_t *restrict *format*, ...);

45435 DESCRIPTION

45436 Refer to *fwprintf()*.

45437 **NAME**

45438 swscanf — convert formatted wide-character input

45439 **SYNOPSIS**

45440 #include <stdio.h>

45441 #include <wchar.h>

```
45442 int swscanf(const wchar_t *restrict ws,  
45443             const wchar_t *restrict format, ... );
```

45444 **DESCRIPTION**45445 Refer to *fwscanf()*.

45446 **NAME**45447 `symlink` — make a symbolic link to a file45448 **SYNOPSIS**45449 `#include <unistd.h>`45450 `int symlink(const char *path1, const char *path2);`45451 **DESCRIPTION**

45452 The `symlink()` function shall create a symbolic link called *path2* that contains the string pointed
 45453 to by *path1* (*path2* is the name of the symbolic link created, *path1* is the string contained in the
 45454 symbolic link).

45455 The string pointed to by *path1* shall be treated only as a character string and shall not be
 45456 validated as a pathname.

45457 If the `symlink()` function fails for any reason other than [EIO], any file named by *path2* shall be
 45458 unaffected.

45459 **RETURN VALUE**

45460 Upon successful completion, `symlink()` shall return 0; otherwise, it shall return `-1` and set *errno* to
 45461 indicate the error.

45462 **ERRORS**45463 The `symlink()` function shall fail if:

45464 [EACCES] Write permission is denied in the directory where the symbolic link is being
 45465 created, or search permission is denied for a component of the path prefix of
 45466 *path2*.

45467 [EEXIST] The *path2* argument names an existing file or symbolic link.

45468 [EIO] An I/O error occurs while reading from or writing to the file system.

45469 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path2*
 45470 argument.

45471 [ENAMETOOLONG]

45472 The length of the *path2* argument exceeds {PATH_MAX} or a pathname
 45473 component is longer than {NAME_MAX} or the length of the *path1* argument
 45474 is longer than {SYMLINK_MAX}.

45475 [ENOENT] A component of *path2* does not name an existing file or *path2* is an empty
 45476 string.

45477 [ENOSPC] The directory in which the entry for the new symbolic link is being placed
 45478 cannot be extended because no space is left on the file system containing the
 45479 directory, or the new symbolic link cannot be created because no space is left
 45480 on the file system which shall contain the link, or the file system is out of file-
 45481 allocation resources.

45482 [ENOTDIR] A component of the path prefix of *path2* is not a directory.

45483 [EROFS] The new symbolic link would reside on a read-only file system.

45484 The `symlink()` function may fail if:

45485 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 45486 resolution of the *path2* argument.

45487 [ENAMETOOLONG]

45488 As a result of encountering a symbolic link in resolution of the *path2*

45489 argument, the length of the substituted pathname string exceeded
45490 {PATH_MAX} bytes (including the terminating null byte), or the length of the
45491 string pointed to by *path1* exceeded {SYMLINK_MAX}.

45492 EXAMPLES

45493 None.

45494 APPLICATION USAGE

45495 Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a
45496 hard link guarantees the existence of a file, even after the original name has been removed. A
45497 symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not
45498 exist when the link is created. A symbolic link can cross file system boundaries.

45499 Normal permission checks are made on each component of the symbolic link pathname during
45500 its resolution.

45501 RATIONALE

45502 Since IEEE Std 1003.1-2001 does not require any association of file times with symbolic links,
45503 there is no requirement that file times be updated by *symlink()*.

45504 FUTURE DIRECTIONS

45505 None.

45506 SEE ALSO

45507 *lchown()*, *link()*, *lstat()*, *open()*, *readlink()*, *unlink()*, the Base Definitions volume of
45508 IEEE Std 1003.1-2001, <**unistd.h**>

45509 CHANGE HISTORY

45510 First released in Issue 4, Version 2.

45511 Issue 5

45512 Moved from X/OPEN UNIX extension to BASE.

45513 Issue 6

45514 The following changes were made to align with the IEEE P1003.1a draft standard:

- 45515 • The DESCRIPTION text is updated.
- 45516 • The [ELOOP] optional error condition is added.

45517 **NAME**

45518 sync — schedule file system updates

45519 **SYNOPSIS**45520 xSI `#include <unistd.h>`45521 `void sync(void);`

45522

45523 **DESCRIPTION**45524 The `sync()` function shall cause all information in memory that updates file systems to be
45525 scheduled for writing out to all file systems.45526 The writing, although scheduled, is not necessarily complete upon return from `sync()`.45527 **RETURN VALUE**45528 The `sync()` function shall not return a value.45529 **ERRORS**

45530 No errors are defined.

45531 **EXAMPLES**

45532 None.

45533 **APPLICATION USAGE**

45534 None.

45535 **RATIONALE**

45536 None.

45537 **FUTURE DIRECTIONS**

45538 None.

45539 **SEE ALSO**45540 `fsync()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<unistd.h>`45541 **CHANGE HISTORY**

45542 First released in Issue 4, Version 2.

45543 **Issue 5**

45544 Moved from X/OPEN UNIX extension to BASE.

45545 **NAME**

45546 sysconf — get configurable system variables

45547 **SYNOPSIS**

45548 #include <unistd.h>

45549 long sysconf(int name);

45550 **DESCRIPTION**

45551 The *sysconf()* function provides a method for the application to determine the current value of a
 45552 configurable system limit or option (*variable*). The implementation shall support all of the
 45553 variables listed in the following table and may support others.

45554 The *name* argument represents the system variable to be queried. The following table lists the
 45555 minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*,
 45556 and the symbolic constants defined in <unistd.h> that are the corresponding values used for
 45557 *name*.

45558	Variable	Value of Name
45560	{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
45561	{AIO_MAX}	_SC_AIO_MAX
45562	{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
45563	{ARG_MAX}	_SC_ARG_MAX
45564	{ATEXIT_MAX}	_SC_ATEXIT_MAX
45565	{BC_BASE_MAX}	_SC_BC_BASE_MAX
45566	{BC_DIM_MAX}	_SC_BC_DIM_MAX
45567	{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
45568	{BC_STRING_MAX}	_SC_BC_STRING_MAX
45569	{CHILD_MAX}	_SC_CHILD_MAX
45570	Clock ticks/second	_SC_CLK_TCK
45571	{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
45572	{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
45573	{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
45574	{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
45575	{IOV_MAX}	_SC_IOV_MAX
45576	{LINE_MAX}	_SC_LINE_MAX
45577	{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
45578	{NGROUPS_MAX}	_SC_NGROUPS_MAX
45579	Maximum size of <i>getgrgid_r()</i> and	_SC_GETGR_R_SIZE_MAX
45580	<i>getgrnam_r()</i> data buffers	
45581	Maximum size of <i>getpwuid_r()</i> and	_SC_GETPW_R_SIZE_MAX
45582	<i>getpwnam_r()</i> data buffers	
45583	{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
45584	{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
45585	{OPEN_MAX}	_SC_OPEN_MAX
45586	_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
45587	_POSIX_BARRIERS	_SC_BARRIERS
45588	_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
45589	_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
45590	_POSIX_CPUTIME	_SC_CPUTIME

45591

45592

45593

45594

45595

45596

45597

45598

45599

45600

45601

45602

45603

45604

45605

45606

45607

45608

45609

45610

45611

45612

45613

45614

45615

45616

45617

45618

45619

45620

45621

45622

45623

45624

45625

45626

45627

45628

45629

45630

45631

45632

45633

45634

45635

45636

45637

45638

45639

Variable	Value of Name
_POSIX_FILE_LOCKING	_SC_FILE_LOCKING
_POSIX_FSYNC	_SC_FSYNC
_POSIX_IPV6	_SC_IPV6
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL
_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
_POSIX_MEMLOCK	_SC_MEMLOCK
_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
_POSIX_MULTI_PROCESS	_SC_MULTI_PROCESS
_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
_POSIX_RAW_SOCKETS	_SC_RAW_SOCKETS
_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
_POSIX_REGEX	_SC_REGEX
_POSIX_SAVED_IDS	_SC_SAVED_IDS
_POSIX_SEMAPHORES	_SC_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
_POSIX_SHELL	_SC_SHELL
_POSIX_SPAWN	_SC_SPAWN
_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
_POSIX_SYMLINK_MAX	_SC_SYMLINK_MAX
_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_CPUTIME	_SC_THREAD_CPUTIME
_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED
_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
_POSIX_THREAD_SPORADIC_SERVER	_SC_THREAD_SPORADIC_SERVER
_POSIX_THREADS	_SC_THREADS
_POSIX_TIMEOUTS	_SC_TIMEOUTS
_POSIX_TIMERS	_SC_TIMERS
_POSIX_TRACE	_SC_TRACE
_POSIX_TRACE_EVENT_FILTER	_SC_TRACE_EVENT_FILTER
_POSIX_TRACE_INHERIT	_SC_TRACE_INHERIT
_POSIX_TRACE_LOG	_SC_TRACE_LOG
_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
_POSIX_VERSION	_SC_VERSION
_POSIX_V6_ILP32_OFF32	_SC_V6_ILP32_OFF32
_POSIX_V6_ILP32_OFFBIG	_SC_V6_ILP32_OFFBIG
_POSIX_V6_LP64_OFF64	_SC_V6_LP64_OFF64

	Variable	Value of Name
45640		
45641		
45642	_POSIX_V6_LPBIG_OFFBIG	_SC_V6_LPBIG_OFFBIG
45643	_POSIX2_C_BIND	_SC_2_C_BIND
45644	_POSIX2_C_DEV	_SC_2_C_DEV
45645	_POSIX2_C_VERSION	_SC_2_C_VERSION
45646	_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM
45647	_POSIX2_FORT_DEV	_SC_2_FORT_DEV
45648	_POSIX2_FORT_RUN	_SC_2_FORT_RUN
45649	_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF
45650	_POSIX2_PBS	_SC_2_PBS
45651	_POSIX2_PBS_ACCOUNTING	_SC_2_PBS_ACCOUNTING
45652	_POSIX2_PBS_CHECKPOINT	_SC_2_PBS_CHECKPOINT
45653	_POSIX2_PBS_LOCATE	_SC_2_PBS_LOCATE
45654	_POSIX2_PBS_MESSAGE	_SC_2_PBS_MESSAGE
45655	_POSIX2_PBS_TRACK	_SC_2_PBS_TRACK
45656	_POSIX2_SW_DEV	_SC_2_SW_DEV
45657	_POSIX2_UPE	_SC_2_UPE
45658	_POSIX2_VERSION	_SC_2_VERSION
45659	_REGEX_VERSION	_SC_REGEX_VERSION
45660	{PAGE_SIZE}	_SC_PAGE_SIZE
45661	{PAGESIZE}	_SC_PAGESIZE
45662	{PTHREAD_DESTRUCTOR_ITERATIONS}	_SC_THREAD_DESTRUCTOR_ITERATIONS
45663	{PTHREAD_KEYS_MAX}	_SC_THREAD_KEYS_MAX
45664	{PTHREAD_STACK_MIN}	_SC_THREAD_STACK_MIN
45665	{PTHREAD_THREADS_MAX}	_SC_THREAD_THREADS_MAX
45666	{RE_DUP_MAX}	_SC_RE_DUP_MAX
45667	{RTSIG_MAX}	_SC_RTSIG_MAX
45668	{SEM_NSEMS_MAX}	_SC_SEM_NSEMS_MAX
45669	{SEM_VALUE_MAX}	_SC_SEM_VALUE_MAX
45670	{SIGQUEUE_MAX}	_SC_SIGQUEUE_MAX
45671	{STREAM_MAX}	_SC_STREAM_MAX
45672	{SYMLOOP_MAX}	_SC_SYMLOOP_MAX
45673	{TIMER_MAX}	_SC_TIMER_MAX
45674	{TTY_NAME_MAX}	_SC_TTY_NAME_MAX
45675	{TZNAME_MAX}	_SC_TZNAME_MAX
45676	_XBS5_ILP32_OFF32 (LEGACY)	_SC_XBS5_ILP32_OFF32 (LEGACY)
45677	_XBS5_ILP32_OFFBIG (LEGACY)	_SC_XBS5_ILP32_OFFBIG (LEGACY)
45678	_XBS5_LP64_OFF64 (LEGACY)	_SC_XBS5_LP64_OFF64 (LEGACY)
45679	_XBS5_LPBIG_OFFBIG (LEGACY)	_SC_XBS5_LPBIG_OFFBIG (LEGACY)
45680	_XOPEN_CRYPT	_SC_XOPEN_CRYPT
45681	_XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N
45682	_XOPEN_LEGACY	_SC_XOPEN_LEGACY
45683	_XOPEN_REALTIME	_SC_XOPEN_REALTIME
45684	_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
45685	_XOPEN_SHM	_SC_XOPEN_SHM
45686	_XOPEN_STREAMS	_SC_XOPEN_STREAMS
45687	_XOPEN_UNIX	_SC_XOPEN_UNIX
45688	_XOPEN_VERSION	_SC_XOPEN_VERSION

45689

45690

45691

Variable	Value of Name
_XOPEN_XCU_VERSION	_SC_XOPEN_XCU_VERSION

45692 **RETURN VALUE**

45693

45694

45695

If *name* is an invalid value, `sysconf()` shall return `-1` and set *errno* to indicate the error. If the variable corresponding to *name* has no limit, `sysconf()` shall return `-1` without changing the value of *errno*. Note that indefinite limits do not imply infinite limits; see **<limits.h>**.

45696

45697

45698

45699 XSI

45700

45701

Otherwise, `sysconf()` shall return the current variable value on the system. The value returned shall not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's **<limits.h>** or **<unistd.h>**. The value shall not change during the lifetime of the calling process, except that `sysconf(_SC_OPEN_MAX)` may return different values before and after a call to `setrlimit()` which changes the `RLIMIT_NOFILE` soft limit.

45702 **ERRORS**

45703

The `sysconf()` function shall fail if:

45704

[EINVAL] The value of the *name* argument is invalid.

45705 **EXAMPLES**

45706

None.

45707 **APPLICATION USAGE**

45708

45709

45710

As `-1` is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call `sysconf()`, and, if it returns `-1`, check to see if *errno* is non-zero.

45711

45712

45713

45714

45715

45716

45717

45718

If the value of `sysconf(_SC_2_VERSION)` is not equal to the value of the `_POSIX2_VERSION` symbolic constant, the utilities available via `system()` or `popen()` might not behave as described in the Shell and Utilities volume of IEEE Std 1003.1-2001. This would mean that the application is not running in an environment that conforms to the Shell and Utilities volume of IEEE Std 1003.1-2001. Some applications might be able to deal with this, others might not. However, the functions defined in this volume of IEEE Std 1003.1-2001 continue to operate as specified, even if `sysconf(_SC_2_VERSION)` reports that the utilities no longer perform as specified.

45719 **RATIONALE**

45720

45721

45722

This functionality was added in response to requirements of application developers and of system vendors who deal with many international system configurations. It is closely related to `pathconf()` and `fpathconf()`.

45723

45724

45725

45726

45727

Although a conforming application can run on all systems by never demanding more resources than the minimum values published in this volume of IEEE Std 1003.1-2001, it is useful for that application to be able to use the actual value for the quantity of a resource available on any given system. To do this, the application makes use of the value of a symbolic constant in **<limits.h>** or **<unistd.h>**.

45728

45729

45730

However, once compiled, the application must still be able to cope if the amount of resource available is increased. To that end, an application may need a means of determining the quantity of a resource, or the presence of an option, at execution time.

45731

Two examples are offered:

45732

45733

45734

1. Applications may wish to act differently on systems with or without job control. Applications vendors who wish to distribute only a single binary package to all instances of a computer architecture would be forced to assume job control is never available if it

were to rely solely on the `<unistd.h>` value published in this volume of IEEE Std 1003.1-2001.

2. International applications vendors occasionally require knowledge of the number of clock ticks per second. Without these facilities, they would be required to either distribute their applications partially in source form or to have 50 Hz and 60 Hz versions for the various countries in which they operate.

It is the knowledge that many applications are actually distributed widely in executable form that leads to this facility. If limited to the most restrictive values in the headers, such applications would have to be prepared to accept the most limited environments offered by the smallest microcomputers. Although this is entirely portable, there was a consensus that they should be able to take advantage of the facilities offered by large systems, without the restrictions associated with source and object distributions.

During the discussions of this feature, it was pointed out that it is almost always possible for an application to discern what a value might be at runtime by suitably testing the various functions themselves. And, in any event, it could always be written to adequately deal with error returns from the various functions. In the end, it was felt that this imposed an unreasonable level of complication and sophistication on the application writer.

This runtime facility is not meant to provide ever-changing values that applications have to check multiple times. The values are seen as changing no more frequently than once per system initialization, such as by a system administrator or operator with an automatic configuration program. This volume of IEEE Std 1003.1-2001 specifies that they shall not change within the lifetime of the process.

Some values apply to the system overall and others vary at the file system or directory level. The latter are described in *pathconf()*.

Note that all values returned must be expressible as integers. String values were considered, but the additional flexibility of this approach was rejected due to its added complexity of implementation and use.

Some values, such as `{PATH_MAX}`, are sometimes so large that they must not be used to, say, allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic constant is not even defined in this case.

Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is infinite, returning an error indicating that some other resource limit has been reached is conforming behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

confstr(), *pathconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<limits.h>`, `<unistd.h>`, the Shell and Utilities volume of IEEE Std 1003.1-2001, *getconf*

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The `_XBS_` variables and name values are added to the table of system variables in the DESCRIPTION. These are all marked EX.

45780 **Issue 6**

45781 The symbol CLK_TCK is obsolescent and removed. It is replaced with the phrase “clock ticks
45782 per second”.

45783 The symbol {PASS_MAX} is removed.

45784 The following changes were made to align with the IEEE P1003.1a draft standard:

- 45785 • Table entries are added for the following variables: _SC_REGEX, _SC_SHELL,
45786 _SC_REGEX_VERSION, _SC_SYMLINK_MAX.

45787 The following `sysconf()` variables and their associated names are added for alignment with
45788 IEEE Std 1003.1d-1999:

```
45789     _POSIX_ADVISORY_INFO
45790     _POSIX_CPUTIME
45791     _POSIX_SPAWN
45792     _POSIX_SPORADIC_SERVER
45793     _POSIX_THREAD_CPUTIME
45794     _POSIX_THREAD_SPORADIC_SERVER
45795     _POSIX_TIMEOUTS
```

45796 The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

- 45797 • A statement expressing the dependency of support for some system variables on
45798 implementation options is added.
- 45799 • The following system variables are added:

```
45800     _POSIX_BARRIERS
45801     _POSIX_CLOCK_SELECTION
45802     _POSIX_MONOTONIC_CLOCK
45803     _POSIX_READER_WRITER_LOCKS
45804     _POSIX_SPIN_LOCKS
45805     _POSIX_TYPED_MEMORY_OBJECTS
```

45806 The following system variables are added for alignment with IEEE Std 1003.2d-1994:

```
45807     _POSIX2_PBS
45808     _POSIX2_PBS_ACCOUNTING
45809     _POSIX2_PBS_LOCATE
45810     _POSIX2_PBS_MESSAGE
45811     _POSIX2_PBS_TRACK
```

45812 The following `sysconf()` variables and their associated names are added for alignment with
45813 IEEE Std 1003.1q-2000:

```
45814     _POSIX_TRACE
45815     _POSIX_TRACE_EVENT_FILTER
45816     _POSIX_TRACE_INHERIT
45817     _POSIX_TRACE_LOG
```

45818 The macros associated with the `c89` programming models are marked `LEGACY`, and new
45819 equivalent macros associated with `c99` are introduced.

45820 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the
45821 DESCRIPTION to denote that the `_PC*` and `_SC*` symbols are now required to be supported. A
45822 corresponding change has been made in the Base Definitions volume of IEEE Std 1003.1-2001.
45823 The deletion in the second paragraph removes some duplicated text. Additional symbols that
45824 were erroneously omitted from this reference page have been added.

45825 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the
45826 RETURN VALUE section that the value returned for `sysconf(_SC_OPEN_MAX)` may change if a
45827 call to `setrlimit()` adjusts the `RLIMIT_NOFILE` soft limit.

45828 NAME

45829 syslog — log a message

45830 SYNOPSIS

45831 xSI #include <syslog.h>

45832 void syslog(int *priority*, const char **message*, ... /* *argument* */);

45833

45834 DESCRIPTION

45835 Refer to *closelog()*.

45836 **NAME**

45837 system — issue a command

45838 **SYNOPSIS**

45839 #include <stdlib.h>

45840 int system(const char **command*);45841 **DESCRIPTION**

45842 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 45843 conflict between the requirements described here and the ISO C standard is unintentional. This
 45844 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45845 If *command* is a null pointer, the *system()* function shall determine whether the host environment
 45846 has a command processor. If *command* is not a null pointer, the *system()* function shall pass the
 45847 string pointed to by *command* to that command processor to be executed in an implementation-
 45848 defined manner; this might then cause the program calling *system()* to behave in a non-
 45849 conforming manner or to terminate.

45850 CX The environment of the executed command shall be as if a child process were created using
 45851 *fork()*, and the child process invoked the *sh* utility using *execl()* as follows:

45852 *execl*(<*shell path*>, "sh", "-c", *command*, (char *)0);45853 where <*shell path*> is an unspecified pathname for the *sh* utility.

45854 The *system()* function shall ignore the SIGINT and SIGQUIT signals, and shall block the
 45855 SIGCHLD signal, while waiting for the command to terminate. If this might cause the
 45856 application to miss a signal that would have killed it, then the application should examine the
 45857 return value from *system()* and take whatever action is appropriate to the application if the
 45858 command terminated due to receipt of a signal.

45859 The *system()* function shall not affect the termination status of any child of the calling processes
 45860 other than the process or processes it itself creates.

45861 The *system()* function shall not return until the child process has terminated.45862 **RETURN VALUE**

45863 If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor
 45864 CX is available, or zero if none is available. The *system()* function shall always return non-zero when
 45865 *command* is NULL.

45866 CX If *command* is not a null pointer, *system()* shall return the termination status of the command
 45867 language interpreter in the format specified by *waitpid()*. The termination status shall be as
 45868 defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents
 45869 the command language interpreter from executing after the child process is created, the return
 45870 value from *system()* shall be as if the command language interpreter had terminated using
 45871 *exit*(127) or *_exit*(127). If a child process cannot be created, or if the termination status for the
 45872 command language interpreter cannot be obtained, *system()* shall return -1 and set *errno* to
 45873 indicate the error.

45874 **ERRORS**45875 CX The *system()* function may set *errno* values as described by *fork()*.45876 In addition, *system()* may fail if:45877 CX [ECHILD] The status of the child process created by *system()* is no longer available.

45878 **EXAMPLES**

45879 None.

45880 **APPLICATION USAGE**

45881 If the return value of `system()` is not `-1`, its value can be decoded through the use of the macros
 45882 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.

45883 Note that, while `system()` must ignore `SIGINT` and `SIGQUIT` and block `SIGCHLD` while waiting
 45884 for the child to terminate, the handling of signals in the executed command is as specified by
 45885 `fork()` and `exec`. For example, if `SIGINT` is being caught or is set to `SIG_DFL` when `system()` is
 45886 called, then the child is started with `SIGINT` handling set to `SIG_DFL`.

45887 Ignoring `SIGINT` and `SIGQUIT` in the parent process prevents coordination problems (two
 45888 processes reading from the same terminal, for example) when the executed command ignores or
 45889 catches one of the signals. It is also usually the correct action when the user has given a
 45890 command to the application to be executed synchronously (as in the `'!'` command in many
 45891 interactive applications). In either case, the signal should be delivered only to the child process,
 45892 not to the application itself. There is one situation where ignoring the signals might have less
 45893 than the desired effect. This is when the application uses `system()` to perform some task invisible
 45894 to the user. If the user typed the interrupt character ("`^C`", for example) while `system()` is being
 45895 used in this way, one would expect the application to be killed, but only the executed command
 45896 is killed. Applications that use `system()` in this way should carefully check the return status from
 45897 `system()` to see if the executed command was successful, and should take appropriate action
 45898 when the command fails.

45899 Blocking `SIGCHLD` while waiting for the child to terminate prevents the application from
 45900 catching the signal and obtaining status from `system()`'s child process before `system()` can get the
 45901 status itself.

45902 The context in which the utility is ultimately executed may differ from that in which `system()`
 45903 was called. For example, file descriptors that have the `FD_CLOEXEC` flag set are closed, and the
 45904 process ID and parent process ID are different. Also, if the executed utility changes its
 45905 environment variables or its current working directory, that change is not reflected in the caller's
 45906 context.

45907 There is no defined way for an application to find the specific path for the shell. However,
 45908 `confstr()` can provide a value for `PATH` that is guaranteed to find the `sh` utility.

45909 **RATIONALE**

45910 The `system()` function should not be used by programs that have set user (or group) ID
 45911 privileges. The `fork()` and `exec` family of functions (except `execlp()` and `execvp()`), should be used
 45912 instead. This prevents any unforeseen manipulation of the environment of the user that could
 45913 cause execution of commands not anticipated by the calling program.

45914 There are three levels of specification for the `system()` function. The ISO C standard gives the
 45915 most basic. It requires that the function exists, and defines a way for an application to query
 45916 whether a command language interpreter exists. It says nothing about the command language or
 45917 the environment in which the command is interpreted.

45918 IEEE Std 1003.1-2001 places additional restrictions on `system()`. It requires that if there is a
 45919 command language interpreter, the environment must be as specified by `fork()` and `exec`. This
 45920 ensures, for example, that close-on-exec works, that file locks are not inherited, and that the
 45921 process ID is different. It also specifies the return value from `system()` when the command line
 45922 can be run, thus giving the application some information about the command's completion
 45923 status.

Finally, IEEE Std 1003.1-2001 requires the command to be interpreted as in the shell command language defined in the Shell and Utilities volume of IEEE Std 1003.1-2001.

Note that, `system(NULL)` is required to return non-zero, indicating that there is a command language interpreter. At first glance, this would seem to conflict with the ISO C standard which allows `system(NULL)` to return zero. There is no conflict, however. A system must have a command language interpreter, and is non-conforming if none is present. It is therefore permissible for the `system()` function on such a system to implement the behavior specified by the ISO C standard as long as it is understood that the implementation does not conform to IEEE Std 1003.1-2001 if `system(NULL)` returns zero.

It was explicitly decided that when `command` is `NULL`, `system()` should not be required to check to make sure that the command language interpreter actually exists with the correct mode, that there are enough processes to execute it, and so on. The call `system(NULL)` could, theoretically, check for such problems as too many existing child processes, and return zero. However, it would be inappropriate to return zero due to such a (presumably) transient condition. If some condition exists that is not under the control of this application and that would cause any `system()` call to fail, that system has been rendered non-conforming.

Early drafts required, or allowed, `system()` to return with `errno` set to `[EINTR]` if it was interrupted with a signal. This error return was removed, and a requirement that `system()` not return until the child has terminated was added. This means that if a `waitpid()` call in `system()` exits with `errno` set to `[EINTR]`, `system()` must reissue the `waitpid()`. This change was made for two reasons:

1. There is no way for an application to clean up if `system()` returns `[EINTR]`, short of calling `wait()`, and that could have the undesirable effect of returning the status of children other than the one started by `system()`.
2. While it might require a change in some historical implementations, those implementations already have to be changed because they use `wait()` instead of `waitpid()`.

Note that if the application is catching `SIGCHLD` signals, it will receive such a signal before a successful `system()` call returns.

To conform to IEEE Std 1003.1-2001, `system()` must use `waitpid()`, or some similar function, instead of `wait()`.

The following code sample illustrates how `system()` might be implemented on an implementation conforming to IEEE Std 1003.1-2001.

```
#include <signal.h>
int system(const char *cmd)
{
    int stat;
    pid_t pid;
    struct sigaction sa, savintr, savequit;
    sigset_t saveblock;
    if (cmd == NULL)
        return(1);
    sa.sa_handler = SIG_IGN;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigemptyset(&savintr.sa_mask);
    sigemptyset(&savequit.sa_mask);
    sigaction(SIGINT, &sa, &savintr);
    sigaction(SIGQUIT, &sa, &savequit);
```

```

45972     sigaddset(&sa.sa_mask, SIGCHLD);
45973     sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
45974     if ((pid = fork()) == 0) {
45975         sigaction(SIGINT, &saveintr, (struct sigaction *)0);
45976         sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
45977         sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
45978         execl("/bin/sh", "sh", "-c", cmd, (char *)0);
45979         _exit(127);
45980     }
45981     if (pid == -1) {
45982         stat = -1; /* errno comes from fork() */
45983     } else {
45984         while (waitpid(pid, &stat, 0) == -1) {
45985             if (errno != EINTR) {
45986                 stat = -1;
45987                 break;
45988             }
45989         }
45990     }
45991     sigaction(SIGINT, &saveintr, (struct sigaction *)0);
45992     sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
45993     sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
45994     return(stat);
45995 }

```

45996 Note that, while a particular implementation of *system()* (such as the one above) can assume a
 45997 particular path for the shell, such a path is not necessarily valid on another system. The above
 45998 example is not portable, and is not intended to be.

45999 One reviewer suggested that an implementation of *system()* might want to use an environment
 46000 variable such as *SHELL* to determine which command interpreter to use. The supposed
 46001 implementation would use the default command interpreter if the one specified by the
 46002 environment variable was not available. This would allow a user, when using an application
 46003 that prompts for command lines to be processed using *system()*, to specify a different command
 46004 interpreter. Such an implementation is discouraged. If the alternate command interpreter did not
 46005 follow the command line syntax specified in the Shell and Utilities volume of
 46006 IEEE Std 1003.1-2001, then changing *SHELL* would render *system()* non-conforming. This would
 46007 affect applications that expected the specified behavior from *system()*, and since the Shell and
 46008 Utilities volume of IEEE Std 1003.1-2001 does not mention that *SHELL* affects *system()*, the
 46009 application would not know that it needed to unset *SHELL*.

46010 FUTURE DIRECTIONS

46011 None.

46012 SEE ALSO

46013 *exec*, *pipe()*, *waitpid()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**,
 46014 **<signal.h>**, **<stdlib.h>**, **<sys/wait.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001, *sh*

46015 CHANGE HISTORY

46016 First released in Issue 1. Derived from Issue 1 of the SVID.

46017 Issue 6

46018 Extensions beyond the ISO C standard are marked.

46019 **NAME**

46020 tan, tanf, tanl — tangent function

46021 **SYNOPSIS**

46022 #include <math.h>

46023 double tan(double x);

46024 float tanf(float x);

46025 long double tanl(long double x);

46026 **DESCRIPTION**

46027 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 46028 conflict between the requirements described here and the ISO C standard is unintentional. This
 46029 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

46030 These functions shall compute the tangent of their argument *x*, measured in radians.

46031 An application wishing to check for error situations should set *errno* to zero and call
 46032 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 46033 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 46034 zero, an error has occurred.

46035 **RETURN VALUE**

46036 Upon successful completion, these functions shall return the tangent of *x*.

46037 If the correct value would cause underflow, and is not representable, a range error may occur,
 46038 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

46039 MX If *x* is NaN, a NaN shall be returned.

46040 If *x* is ± 0 , *x* shall be returned.

46041 If *x* is subnormal, a range error may occur and *x* should be returned.

46042 If *x* is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 46043 defined value shall be returned.

46044 If the correct value would cause underflow, and is representable, a range error may occur and
 46045 the correct value shall be returned.

46046 XSI If the correct value would cause overflow, a range error shall occur and *tan()*, *tanf()*, and *tanl()*
 46047 shall return $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and $\pm \text{HUGE\_VALL}$, respectively, with the same sign
 46048 as the correct value of the function.

46049 **ERRORS**

46050 These functions shall fail if:

46051 MX **Domain Error** The value of *x* is $\pm \text{Inf}$.

46052 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 46053 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 46054 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 46055 shall be raised.

46056 XSI **Range Error** The result overflows

46057 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 46058 then *errno* shall be set to [ERANGE]. If the integer expression
 46059 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 46060 floating-point exception shall be raised.

46061 These functions may fail if:

46062 MX Range Error The result underflows, or the value of x is subnormal.

46063 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 46064 then *errno* shall be set to [ERANGE]. If the integer expression
 46065 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 46066 floating-point exception shall be raised.

46067 EXAMPLES

46068 Taking the Tangent of a 45-Degree Angle

```
46069 #include <math.h>
46070 ...
46071 double radians = 45.0 * M_PI / 180;
46072 double result;
46073 ...
46074 result = tan (radians);
```

46075 APPLICATION USAGE

46076 There are no known floating-point representations such that for a normal argument, $\tan(x)$ is
 46077 either overflow or underflow.

46078 These functions may lose accuracy when their argument is near a multiple of $\pi/2$ or is far from
 46079 0.0.

46080 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 46081 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46082 RATIONALE

46083 None.

46084 FUTURE DIRECTIONS

46085 None.

46086 SEE ALSO

46087 *atan()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 46088 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

46089 CHANGE HISTORY

46090 First released in Issue 1. Derived from Issue 1 of the SVID.

46091 Issue 5

46092 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
 46093 in previous issues.

46094 Issue 6

46095 The *tanf()* and *tanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46096 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 46097 revised to align with the ISO/IEC 9899:1999 standard.

46098 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 46099 marked.

46100 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last
 46101 paragraph in the RETURN VALUE section.

46102 **NAME**

46103 tanh, tanhf, tanhl — hyperbolic tangent functions

46104 **SYNOPSIS**

46105 #include <math.h>

46106 double tanh(double x);

46107 float tanhf(float x);

46108 long double tanhl(long double x);

46109 **DESCRIPTION**

46110 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 46111 conflict between the requirements described here and the ISO C standard is unintentional. This
 46112 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

46113 These functions shall compute the hyperbolic tangent of their argument *x*.

46114 An application wishing to check for error situations should set *errno* to zero and call
 46115 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 46116 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 46117 zero, an error has occurred.

46118 **RETURN VALUE**46119 Upon successful completion, these functions shall return the hyperbolic tangent of *x*.46120 MX If *x* is NaN, a NaN shall be returned.46121 If *x* is ± 0 , *x* shall be returned.46122 If *x* is $\pm \text{Inf}$, ± 1 shall be returned.46123 If *x* is subnormal, a range error may occur and *x* should be returned.46124 **ERRORS**

46125 These functions may fail if:

46126 MX Range Error The value of *x* is subnormal.

46127 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 46128 then *errno* shall be set to [ERANGE]. If the integer expression
 46129 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 46130 floating-point exception shall be raised.

46131 **EXAMPLES**

46132 None.

46133 **APPLICATION USAGE**

46134 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 46135 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46136 **RATIONALE**

46137 None.

46138 **FUTURE DIRECTIONS**

46139 None.

46140 **SEE ALSO**

46141 *atanh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*, the Base Definitions volume of
 46142 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
 46143 <math.h>

46144 **CHANGE HISTORY**

46145 First released in Issue 1. Derived from Issue 1 of the SVID.

46146 **Issue 5**

46147 The DESCRIPTION is updated to indicate how an application should check for an error. This
46148 text was previously published in the APPLICATION USAGE section.

46149 **Issue 6**

46150 The *tanhf()* and *tanhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46151 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
46152 revised to align with the ISO/IEC 9899:1999 standard.

46153 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
46154 marked.

46155 **NAME**

46156 tanl — tangent function

46157 **SYNOPSIS**

46158 #include <math.h>

46159 long double tanl(long double x);

46160 **DESCRIPTION**46161 Refer to *tan()*.

46162 **NAME**

46163 tcdrain — wait for transmission of output

46164 **SYNOPSIS**

46165 #include <termios.h>

46166 int tcdrain(int *fildev*);46167 **DESCRIPTION**46168 The *tcdrain()* function shall block until all output written to the object referred to by *fildev* is
46169 transmitted. The *fildev* argument is an open file descriptor associated with a terminal.46170 Any attempts to use *tcdrain()* from a process which is a member of a background process group
46171 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a
46172 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process
46173 shall be allowed to perform the operation, and no signal is sent.46174 **RETURN VALUE**46175 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
46176 indicate the error.46177 **ERRORS**46178 The *tcdrain()* function shall fail if:46179 [EBADF] The *fildev* argument is not a valid file descriptor.46180 [EINTR] A signal interrupted *tcdrain()*.46181 [ENOTTY] The file associated with *fildev* is not a terminal.46182 The *tcdrain()* function may fail if:46183 [EIO] The process group of the writing process is orphaned, and the writing process
46184 is not ignoring or blocking SIGTTOU.46185 **EXAMPLES**

46186 None.

46187 **APPLICATION USAGE**

46188 None.

46189 **RATIONALE**

46190 None.

46191 **FUTURE DIRECTIONS**

46192 None.

46193 **SEE ALSO**46194 *tcflush()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal
46195 Interface, <termios.h>, <unistd.h>46196 **CHANGE HISTORY**

46197 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46198 **Issue 6**

46199 The following new requirements on POSIX implementations derive from alignment with the
46200 Single UNIX Specification:

- 46201 • In the DESCRIPTION, the final paragraph is no longer conditional on
46202 _POSIX_JOB_CONTROL. This is a FIPS requirement.
- 46203 • The [EIO] error is added.

46204 **NAME**

46205 tcflow — suspend or restart the transmission or reception of data

46206 **SYNOPSIS**

46207 #include <termios.h>

46208 int tcflow(int *fildes*, int *action*);46209 **DESCRIPTION**

46210 The *tcflow()* function shall suspend or restart transmission or reception of data on the object referred to by *fildes*, depending on the value of *action*. The *fildes* argument is an open file descriptor associated with a terminal.

- 46213 • If *action* is TCOOFF, output shall be suspended.
- 46214 • If *action* is TCOON, suspended output shall be restarted.
- 46215 • If *action* is TCIOFF, the system shall transmit a STOP character, which is intended to cause the terminal device to stop transmitting data to the system.
- 46216 • If *action* is TCION, the system shall transmit a START character, which is intended to cause the terminal device to start transmitting data to the system.

46219 The default on the opening of a terminal file is that neither its input nor its output are suspended.

46221 Attempts to use *tcflow()* from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, shall cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

46225 **RETURN VALUE**

46226 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

46228 **ERRORS**46229 The *tcflow()* function shall fail if:

- 46230 [EBADF] The *fildes* argument is not a valid file descriptor.
- 46231 [EINVAL] The *action* argument is not a supported value.
- 46232 [ENOTTY] The file associated with *fildes* is not a terminal.

46233 The *tcflow()* function may fail if:

- 46234 [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

46236 **EXAMPLES**

46237 None.

46238 **APPLICATION USAGE**

46239 None.

46240 **RATIONALE**

46241 None.

46242 **FUTURE DIRECTIONS**

46243 None.

46244 **SEE ALSO**

46245 *tcsendbreak()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal
46246 Interface, **<termios.h>**, **<unistd.h>**

46247 **CHANGE HISTORY**

46248 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46249 **Issue 6**

46250 The following new requirements on POSIX implementations derive from alignment with the
46251 Single UNIX Specification:

- 46252 • The [EIO] error is added.

46253 **NAME**

46254 tcflush — flush non-transmitted output data, non-read input data, or both

46255 **SYNOPSIS**

46256 #include <termios.h>

46257 int tcflush(int *fildes*, int *queue_selector*);46258 **DESCRIPTION**

46259 Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildes*
 46260 (an open file descriptor associated with a terminal) but not transmitted, or data received but not
 46261 read, depending on the value of *queue_selector*:

- 46262 • If *queue_selector* is TCIFLUSH, it shall flush data received but not read.
- 46263 • If *queue_selector* is TCOFLUSH, it shall flush data written but not transmitted.
- 46264 • If *queue_selector* is TCIOFLUSH, it shall flush both data received but not read and data
 46265 written but not transmitted.

46266 Attempts to use *tcflush()* from a process which is a member of a background process group on a
 46267 *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU
 46268 signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be
 46269 allowed to perform the operation, and no signal is sent.

46270 **RETURN VALUE**

46271 Upon successful completion, 0 shall be returned. Otherwise, –1 shall be returned and *errno* set to
 46272 indicate the error.

46273 **ERRORS**46274 The *tcflush()* function shall fail if:

- 46275 [EBADF] The *fildes* argument is not a valid file descriptor.
- 46276 [EINVAL] The *queue_selector* argument is not a supported value.
- 46277 [ENOTTY] The file associated with *fildes* is not a terminal.

46278 The *tcflush()* function may fail if:

- 46279 [EIO] The process group of the writing process is orphaned, and the writing process
 46280 is not ignoring or blocking SIGTTOU.

46281 **EXAMPLES**

46282 None.

46283 **APPLICATION USAGE**

46284 None.

46285 **RATIONALE**

46286 None.

46287 **FUTURE DIRECTIONS**

46288 None.

46289 **SEE ALSO**

46290 *tcdrain()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal
 46291 Interface, <termios.h>, <unistd.h>

46292 **CHANGE HISTORY**

46293 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46294 **Issue 6**

46295 The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE
46296 sections, references to *tcflow()* are replaced with *tcflush()*.

46297 The following new requirements on POSIX implementations derive from alignment with the
46298 Single UNIX Specification:

- 46299 • In the DESCRIPTION, the final paragraph is no longer conditional on
46300 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.
- 46301 • The [EIO] error is added.

46302 **NAME**

46303 tcgetattr — get the parameters associated with the terminal

46304 **SYNOPSIS**

46305 #include <termios.h>

46306 int tcgetattr(int *fildev*, struct termios **termios_p*);46307 **DESCRIPTION**

46308 The *tcgetattr()* function shall get the parameters associated with the terminal referred to by *fildev*
 46309 and store them in the **termios** structure referenced by *termios_p*. The *fildev* argument is an open
 46310 file descriptor associated with a terminal.

46311 The *termios_p* argument is a pointer to a **termios** structure.

46312 The *tcgetattr()* operation is allowed from any process.

46313 If the terminal device supports different input and output baud rates, the baud rates stored in
 46314 the **termios** structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are
 46315 equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be
 46316 the actual baud rate. If the terminal device does not support split baud rates, the input baud rate
 46317 stored in the **termios** structure shall be the output rate (as one of the symbolic values).

46318 **RETURN VALUE**

46319 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 46320 indicate the error.

46321 **ERRORS**

46322 The *tcgetattr()* function shall fail if:

46323 [EBADF] The *fildev* argument is not a valid file descriptor.

46324 [ENOTTY] The file associated with *fildev* is not a terminal.

46325 **EXAMPLES**

46326 None.

46327 **APPLICATION USAGE**

46328 None.

46329 **RATIONALE**

46330 Care must be taken when changing the terminal attributes. Applications should always do a
 46331 *tcgetattr()*, save the **termios** structure values returned, and then do a *tcsetattr()*, changing only
 46332 the necessary fields. The application should use the values saved from the *tcgetattr()* to reset the
 46333 terminal state whenever it is done with the terminal. This is necessary because terminal
 46334 attributes apply to the underlying port and not to each individual open instance; that is, all
 46335 processes that have used the terminal see the latest attribute changes.

46336 A program that uses these functions should be written to catch all signals and take other
 46337 appropriate actions to ensure that when the program terminates, whether planned or not, the
 46338 terminal device's state is restored to its original state.

46339 Existing practice dealing with error returns when only part of a request can be honored is based
 46340 on calls to the *ioctl()* function. In historical BSD and System V implementations, the
 46341 corresponding *ioctl()* returns zero if the requested actions were semantically correct, even if
 46342 some of the requested changes could not be made. Many existing applications assume this
 46343 behavior and would no longer work correctly if the return value were changed from zero to -1
 46344 in this case.

46345 Note that either specification has a problem. When zero is returned, it implies everything
46346 succeeded even if some of the changes were not made. When -1 is returned, it implies
46347 everything failed even though some of the changes were made.

46348 Applications that need all of the requested changes made to work properly should follow
46349 *tcsetattr()* with a call to *tcgetattr()* and compare the appropriate field values.

46350 **FUTURE DIRECTIONS**

46351 None.

46352 **SEE ALSO**

46353 *tcsetattr()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal
46354 Interface, <**termios.h**>

46355 **CHANGE HISTORY**

46356 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46357 **Issue 6**

46358 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.
46359 Previously, the number zero was also allowed but was obsolescent.

46360 **NAME**

46361 tcgetpgrp — get the foreground process group ID

46362 **SYNOPSIS**

46363 #include <unistd.h>

46364 pid_t tcgetpgrp(int *fildes*);46365 **DESCRIPTION**46366 The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process
46367 group associated with the terminal.46368 If there is no foreground process group, *tcgetpgrp()* shall return a value greater than 1 that does
46369 not match the process group ID of any existing process group.46370 The *tcgetpgrp()* function is allowed from a process that is a member of a background process
46371 group; however, the information may be subsequently changed by a process that is a member of
46372 a foreground process group.46373 **RETURN VALUE**46374 Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the
46375 foreground process associated with the terminal. Otherwise, -1 shall be returned and *errno* set to
46376 indicate the error.46377 **ERRORS**46378 The *tcgetpgrp()* function shall fail if:46379 [EBADF] The *fildes* argument is not a valid file descriptor.46380 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
46381 controlling terminal.46382 **EXAMPLES**

46383 None.

46384 **APPLICATION USAGE**

46385 None.

46386 **RATIONALE**

46387 None.

46388 **FUTURE DIRECTIONS**

46389 None.

46390 **SEE ALSO**46391 *setsid()*, *setpgid()*, *tcsetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-2001,
46392 <sys/types.h>, <unistd.h>46393 **CHANGE HISTORY**

46394 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46395 **Issue 6**

46396 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

46397 The following new requirements on POSIX implementations derive from alignment with the
46398 Single UNIX Specification:

- 46399
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
46400 required for conforming implementations of previous POSIX specifications, it was not
46401 required for UNIX applications.

46402

46403

- In the DESCRIPTION, text previously conditional on support for _POSIX_JOB_CONTROL is now mandatory. This is a FIPS requirement.

46404 **NAME**

46405 tcgetsid — get the process group ID for the session leader for the controlling terminal

46406 **SYNOPSIS**

46407 xSI #include <termios.h>

46408 pid_t tcgetsid(int *fildes*);

46409

46410 **DESCRIPTION**46411 The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal
46412 specified by *fildes* is the controlling terminal.46413 **RETURN VALUE**46414 Upon successful completion, *tcgetsid()* shall return the process group ID associated with the
46415 terminal. Otherwise, a value of (**pid_t**)-1 shall be returned and *errno* set to indicate the error.46416 **ERRORS**46417 The *tcgetsid()* function shall fail if:46418 [EBADF] The *fildes* argument is not a valid file descriptor.46419 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
46420 controlling terminal.46421 **EXAMPLES**

46422 None.

46423 **APPLICATION USAGE**

46424 None.

46425 **RATIONALE**

46426 None.

46427 **FUTURE DIRECTIONS**

46428 None.

46429 **SEE ALSO**46430 The Base Definitions volume of IEEE Std 1003.1-2001, <**termios.h**>46431 **CHANGE HISTORY**

46432 First released in Issue 4, Version 2.

46433 **Issue 5**

46434 Moved from X/OPEN UNIX extension to BASE.

46435 The [EACCES] error has been removed from the list of mandatory errors, and the description of
46436 [ENOTTY] has been reworded.

46437 **NAME**

46438 tcsendbreak — send a break for a specific duration

46439 **SYNOPSIS**

46440 #include <termios.h>

46441 int tcsendbreak(int *fildes*, int *duration*);46442 **DESCRIPTION**

46443 If the terminal is using asynchronous serial data transmission, *tcsendbreak()* shall cause
46444 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it
46445 shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5
46446 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period
46447 of time.

46448 The *fildes* argument is an open file descriptor associated with a terminal.

46449 If the terminal is not using asynchronous serial data transmission, it is implementation-defined
46450 whether *tcsendbreak()* sends data to generate a break condition or returns without taking any
46451 action.

46452 Attempts to use *tcsendbreak()* from a process which is a member of a background process group
46453 on a *fildes* associated with its controlling terminal shall cause the process group to be sent a
46454 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process
46455 shall be allowed to perform the operation, and no signal is sent.

46456 **RETURN VALUE**

46457 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
46458 indicate the error.

46459 **ERRORS**

46460 The *tcsendbreak()* function shall fail if:

46461 [EBADF] The *fildes* argument is not a valid file descriptor.

46462 [ENOTTY] The file associated with *fildes* is not a terminal.

46463 The *tcsendbreak()* function may fail if:

46464 [EIO] The process group of the writing process is orphaned, and the writing process
46465 is not ignoring or blocking SIGTTOU.

46466 **EXAMPLES**

46467 None.

46468 **APPLICATION USAGE**

46469 None.

46470 **RATIONALE**

46471 None.

46472 **FUTURE DIRECTIONS**

46473 None.

46474 **SEE ALSO**

46475 The Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface,
46476 <termios.h>, <unistd.h>

46477 CHANGE HISTORY

46478 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46479 Issue 6

46480 The following new requirements on POSIX implementations derive from alignment with the
46481 Single UNIX Specification:

- 46482 • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now
46483 mandated. This is a FIPS requirement.
- 46484 • The [EIO] error is added.

46485 **NAME**

46486 tcsetattr — set the parameters associated with the terminal

46487 **SYNOPSIS**

46488 #include <termios.h>

```
46489 int tcsetattr(int fildes, int optional_actions,
46490               const struct termios *termios_p);
```

46491 **DESCRIPTION**

46492 The *tcsetattr()* function shall set the parameters associated with the terminal referred to by the
 46493 open file descriptor *fildes* (an open file descriptor associated with a terminal) from the **termios**
 46494 structure referenced by *termios_p* as follows:

- 46495 • If *optional_actions* is TCSANOW, the change shall occur immediately.
- 46496 • If *optional_actions* is TCSADRAIN, the change shall occur after all output written to *fildes* is
 46497 transmitted. This function should be used when changing parameters that affect output.
- 46498 • If *optional_actions* is TCSAFLUSH, the change shall occur after all output written to *fildes* is
 46499 transmitted, and all input so far received but not read shall be discarded before the change is
 46500 made.

46501 If the output baud rate stored in the **termios** structure pointed to by *termios_p* is the zero baud
 46502 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the
 46503 line.

46504 If the input baud rate stored in the **termios** structure pointed to by *termios_p* is 0, the input baud
 46505 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.

46506 The *tcsetattr()* function shall return successfully if it was able to perform any of the requested
 46507 actions, even if some of the requested actions could not be performed. It shall set all the
 46508 attributes that the implementation supports as requested and leave all the attributes not
 46509 supported by the implementation unchanged. If no part of the request can be honored, it shall
 46510 return `-1` and set *errno* to `[EINVAL]`. If the input and output baud rates differ and are a
 46511 combination that is not supported, neither baud rate shall be changed. A subsequent call to
 46512 *tcgetattr()* shall return the actual state of the terminal device (reflecting both the changes made
 46513 and not made in the previous *tcsetattr()* call). The *tcsetattr()* function shall not change the values
 46514 found in the **termios** structure under any circumstances.

46515 The effect of *tcsetattr()* is undefined if the value of the **termios** structure pointed to by *termios_p*
 46516 was not derived from the result of a call to *tcgetattr()* on *fildes*; an application should modify
 46517 only fields and flags defined by this volume of IEEE Std 1003.1-2001 between the call to
 46518 *tcgetattr()* and *tcsetattr()*, leaving all other fields and flags unmodified.

46519 No actions defined by this volume of IEEE Std 1003.1-2001, other than a call to *tcsetattr()* or a
 46520 close of the last file descriptor in the system associated with this terminal device, shall cause any
 46521 of the terminal attributes defined by this volume of IEEE Std 1003.1-2001 to change.

46522 If *tcsetattr()* is called from a process which is a member of a background process group on a
 46523 *fildes* associated with its controlling terminal:

- 46524 • If the calling process is blocking or ignoring SIGTTOU signals, the operation completes
 46525 normally and no signal is sent.
- 46526 • Otherwise, a SIGTTOU signal shall be sent to the process group.

46527 **RETURN VALUE**

46528 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 46529 indicate the error.

46530 **ERRORS**

46531 The *tcsetattr()* function shall fail if:

46532 [EBADF] The *fildev* argument is not a valid file descriptor.

46533 [EINTR] A signal interrupted *tcsetattr()*.

46534 [EINVAL] The *optional_actions* argument is not a supported value, or an attempt was
 46535 made to change an attribute represented in the **termios** structure to an
 46536 unsupported value.

46537 [ENOTTY] The file associated with *fildev* is not a terminal.

46538 The *tcsetattr()* function may fail if:

46539 [EIO] The process group of the writing process is orphaned, and the writing process
 46540 is not ignoring or blocking SIGTTOU.

46541 **EXAMPLES**

46542 None.

46543 **APPLICATION USAGE**

46544 If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to
 46545 determine what baud rates were actually selected.

46546 **RATIONALE**

46547 The *tcsetattr()* function can be interrupted in the following situations:

- 46548 • It is interrupted while waiting for output to drain.
- 46549 • It is called from a process in a background process group and SIGTTOU is caught.

46550 See also the RATIONALE section in *tcgetattr()*.

46551 **FUTURE DIRECTIONS**

46552 Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be
 46553 supported in a future version of this volume of IEEE Std 1003.1-2001.

46554 **SEE ALSO**

46555 *cfgetispeed()*, *tcgetattr()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11,
 46556 General Terminal Interface, **<termios.h>**, **<unistd.h>**

46557 **CHANGE HISTORY**

46558 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46559 **Issue 6**

46560 The following new requirements on POSIX implementations derive from alignment with the
 46561 Single UNIX Specification:

- 46562 • In the DESCRIPTION, text previously conditional on **_POSIX_JOB_CONTROL** is now
 46563 mandated. This is a FIPS requirement.
- 46564 • The [EIO] error is added.

46565 In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of
46566 a background process group is clarified.

46567 **NAME**

46568 tcsetpgrp — set the foreground process group ID

46569 **SYNOPSIS**

46570 #include <unistd.h>

46571 int tcsetpgrp(int *fildev*, pid_t *pgid_id*);46572 **DESCRIPTION**

46573 If the process has a controlling terminal, *tcsetpgrp()* shall set the foreground process group ID
 46574 associated with the terminal to *pgid_id*. The application shall ensure that the file associated with
 46575 *fildev* is the controlling terminal of the calling process and the controlling terminal is currently
 46576 associated with the session of the calling process. The application shall ensure that the value of
 46577 *pgid_id* matches a process group ID of a process in the same session as the calling process.

46578 Attempts to use *tcsetpgrp()* from a process which is a member of a background process group on
 46579 a *fildev* associated with its controlling terminal shall cause the process group to be sent a
 46580 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process
 46581 shall be allowed to perform the operation, and no signal is sent.

46582 **RETURN VALUE**

46583 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 46584 indicate the error.

46585 **ERRORS**46586 The *tcsetpgrp()* function shall fail if:

46587	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor.
46588	[EINVAL]	This implementation does not support the value in the <i>pgid_id</i> argument.
46589	[ENOTTY]	The calling process does not have a controlling terminal, or the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.
46590		
46591		
46592	[EPERM]	The value of <i>pgid_id</i> is a value supported by the implementation, but does not match the process group ID of a process in the same session as the calling process.
46593		
46594		

46595 **EXAMPLES**

46596 None.

46597 **APPLICATION USAGE**

46598 None.

46599 **RATIONALE**

46600 None.

46601 **FUTURE DIRECTIONS**

46602 None.

46603 **SEE ALSO**46604 *tcgetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>46605 **CHANGE HISTORY**

46606 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46607 **Issue 6**

46608 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

46609 The following new requirements on POSIX implementations derive from alignment with the
46610 Single UNIX Specification:

46611 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
46612 required for conforming implementations of previous POSIX specifications, it was not
46613 required for UNIX applications.

46614 • In the DESCRIPTION and ERRORS sections, text previously conditional on
46615 `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

46616 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

46617 The Open Group Corrigendum U047/4 is applied.

46618 **NAME**

46619 tdelete, tfind, tsearch, twalk — manage a binary search tree

46620 **SYNOPSIS**

```

46621 xSI #include <search.h>

46622 void *tdelete(const void *restrict key, void **restrict rootp,
46623             int (*compar)(const void *, const void *));
46624 void *tfind(const void *key, void *const *rootp,
46625            int (*compar)(const void *, const void *));
46626 void *tsearch(const void *key, void **rootp,
46627              int (*compar)(const void *, const void *));
46628 void twalk(const void *root,
46629           void (*action)(const void *, VISIT, int));
46630

```

46631 **DESCRIPTION**

46632 The *tdelete()*, *tfind()*, *tsearch()*, and *twalk()* functions manipulate binary search trees.
 46633 Comparisons are made with a user-supplied routine, the address of which is passed as the
 46634 *compar* argument. This routine is called with two arguments, which are the pointers to the
 46635 elements being compared. The application shall ensure that the user-supplied routine returns an
 46636 integer less than, equal to, or greater than 0, according to whether the first argument is to be
 46637 considered less than, equal to, or greater than the second argument. The comparison function
 46638 need not compare every byte, so arbitrary data may be contained in the elements in addition to
 46639 the values being compared.

46640 The *tsearch()* function shall build and access the tree. The *key* argument is a pointer to an element
 46641 to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed
 46642 to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key*
 46643 shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a
 46644 pointer to this node returned. Only pointers are copied, so the application shall ensure that the
 46645 calling routine stores the data. The *rootp* argument points to a variable that points to the root
 46646 node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree;
 46647 in this case, the variable shall be set to point to the node which shall be at the root of the new
 46648 tree.

46649 Like *tsearch()*, *tfind()* shall search for a node in the tree, returning a pointer to it if found.
 46650 However, if it is not found, *tfind()* shall return a null pointer. The arguments for *tfind()* are the
 46651 same as for *tsearch()*.

46652 The *tdelete()* function shall delete a node from a binary search tree. The arguments are the same
 46653 as for *tsearch()*. The variable pointed to by *rootp* shall be changed if the deleted node was the
 46654 root of the tree. The *tdelete()* function shall return a pointer to the parent of the deleted node, or a
 46655 null pointer if the node is not found.

46656 The *twalk()* function shall traverse a binary search tree. The *root* argument is a pointer to the root
 46657 node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below
 46658 that node.) The argument *action* is the name of a routine to be invoked at each node. This routine
 46659 is, in turn, called with three arguments. The first argument shall be the address of the node being
 46660 visited. The structure pointed to by this argument is unspecified and shall not be modified by
 46661 the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-
 46662 element to access the element stored in the node. The second argument shall be a value from an
 46663 enumeration data type:

```

46664 typedef enum { preorder, postorder, endorder, leaf } VISIT;

```

46665 (defined in **<search.h>**), depending on whether this is the first, second, or third time that the
 46666 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a
 46667 leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

46668 If the calling function alters the pointer to the root, the result is undefined.

46669 RETURN VALUE

46670 If the node is found, both *tsearch()* and *tfind()* shall return a pointer to it. If not, *tfind()* shall
 46671 return a null pointer, and *tsearch()* shall return a pointer to the inserted item.

46672 A null pointer shall be returned by *tsearch()* if there is not enough space available to create a new
 46673 node.

46674 A null pointer shall be returned by *tdelete()*, *tfind()*, and *tsearch()* if *rootp* is a null pointer on
 46675 entry.

46676 The *tdelete()* function shall return a pointer to the parent of the deleted node, or a null pointer if
 46677 the node is not found.

46678 The *twalk()* function shall not return a value.

46679 ERRORS

46680 No errors are defined.

46681 EXAMPLES

46682 The following code reads in strings and stores structures containing a pointer to each string and
 46683 a count of its length. It then walks the tree, printing out the stored strings and their lengths in
 46684 alphabetical order.

```

46685 #include <search.h>
46686 #include <string.h>
46687 #include <stdio.h>

46688 #define STRSZ    10000
46689 #define NODSZ    500

46690 struct node {      /* Pointers to these are stored in the tree. */
46691     char    *string;
46692     int     length;
46693 };

46694 char    string_space[STRSZ]; /* Space to store strings. */
46695 struct node nodes[NODSZ];    /* Nodes to store. */
46696 void    *root = NULL;        /* This points to the root. */

46697 int main(int argc, char *argv[])
46698 {
46699     char    *strptr = string_space;
46700     struct node *nodeptr = nodes;
46701     void    print_node(const void *, VISIT, int);
46702     int     i = 0, node_compare(const void *, const void *);

46703     while (gets(strptr) != NULL && i++ < NODSZ) {
46704         /* Set node. */
46705         nodeptr->string = strptr;
46706         nodeptr->length = strlen(strptr);
46707         /* Put node into the tree. */
46708         (void) tsearch((void *)nodeptr, (void **)&root,
46709             node_compare);

```

```

46710         /* Adjust pointers, so we do not overwrite tree. */
46711         strptr += nodeptr->length + 1;
46712         nodeptr++;
46713     }
46714     twalk(root, print_node);
46715     return 0;
46716 }

46717 /*
46718  * This routine compares two nodes, based on an
46719  * alphabetical ordering of the string field.
46720  */
46721 int
46722 node_compare(const void *node1, const void *node2)
46723 {
46724     return strcmp(((const struct node *) node1)->string,
46725                  ((const struct node *) node2)->string);
46726 }

46727 /*
46728  * This routine prints out a node, the second time
46729  * twalk encounters it or if it is a leaf.
46730  */
46731 void
46732 print_node(const void *ptr, VISIT order, int level)
46733 {
46734     const struct node *p = *(const struct node **) ptr;
46735     if (order == postorder || order == leaf) {
46736         (void) printf("string = %s, length = %d\n",
46737                     p->string, p->length);
46738     }
46739 }

```

46740 APPLICATION USAGE

46741 The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()*
 46742 and *tsearch()*.

46743 There are two nomenclatures used to refer to the order in which tree nodes are visited. The
 46744 *tsearch()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node
 46745 before any of its children, after its left child and before its right, and after both its children. The
 46746 alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which
 46747 could result in some confusion over the meaning of **postorder**.

46748 RATIONALE

46749 None.

46750 FUTURE DIRECTIONS

46751 None.

46752 SEE ALSO

46753 *hcreate()*, *lsearch()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**search.h**>

46754 **CHANGE HISTORY**

46755 First released in Issue 1. Derived from Issue 1 of the SVID.

46756 **Issue 5**

46757 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
46758 previous issues.

46759 **Issue 6**

46760 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

46761 The **restrict** keyword is added to the *tdelete()* prototype for alignment with the
46762 ISO / IEC 9899:1999 standard.

46763 **NAME**

46764 telldir — current location of a named directory stream

46765 **SYNOPSIS**46766 XSI `#include <dirent.h>`46767 `long telldir(DIR *dirp);`

46768

46769 **DESCRIPTION**46770 The *telldir()* function shall obtain the current location associated with the directory stream
46771 specified by *dirp*.46772 If the most recent operation on the directory stream was a *seekdir()*, the directory position
46773 returned from the *telldir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.46774 **RETURN VALUE**46775 Upon successful completion, *telldir()* shall return the current location of the specified directory
46776 stream.46777 **ERRORS**

46778 No errors are defined.

46779 **EXAMPLES**

46780 None.

46781 **APPLICATION USAGE**

46782 None.

46783 **RATIONALE**

46784 None.

46785 **FUTURE DIRECTIONS**

46786 None.

46787 **SEE ALSO**46788 *opendir()*, *readdir()*, *seekdir()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**46789 **CHANGE HISTORY**

46790 First released in Issue 2.

46791 **NAME**

46792 tempnam — create a name for a temporary file

46793 **SYNOPSIS**46794 xSI `#include <stdio.h>`46795 `char *tempnam(const char *dir, const char *pfx);`

46796

46797 **DESCRIPTION**46798 The *tempnam()* function shall generate a pathname that may be used for a temporary file.

46799 The *tempnam()* function allows the user to control the choice of a directory. The *dir* argument
 46800 points to the name of the directory in which the file is to be created. If *dir* is a null pointer or
 46801 points to a string which is not a name for an appropriate directory, the path prefix defined as
 46802 P_tmpdir in the **<stdio.h>** header shall be used. If that directory is not accessible, an
 46803 implementation-defined directory may be used.

46804 Many applications prefer their temporary files to have certain initial letter sequences in their
 46805 names. The *pfx* argument should be used for this. This argument may be a null pointer or point
 46806 to a string of up to five bytes to be used as the beginning of the filename.

46807 Some implementations of *tempnam()* may use *tmpnam()* internally. On such implementations, if
 46808 called more than {TMP_MAX} times in a single process, the behavior is implementation-defined.

46809 **RETURN VALUE**

46810 Upon successful completion, *tempnam()* shall allocate space for a string, put the generated
 46811 pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a
 46812 subsequent call to *free()*. Otherwise, it shall return a null pointer and set *errno* to indicate the
 46813 error.

46814 **ERRORS**46815 The *tempnam()* function shall fail if:

46816 [ENOMEM] Insufficient storage space is available.

46817 **EXAMPLES**46818 **Generating a Pathname**

46819 The following example generates a pathname for a temporary file in directory **/tmp**, with the
 46820 prefix *file*. After the filename has been created, the call to *free()* deallocates the space used to
 46821 store the filename.

```
46822 #include <stdio.h>
46823 #include <stdlib.h>
46824 ...
46825 char *directory = "/tmp";
46826 char *fileprefix = "file";
46827 char *file;

46828 file = tempnam(directory, fileprefix);
46829 free(file);
```

46830 **APPLICATION USAGE**

46831 This function only creates pathnames. It is the application's responsibility to create and remove
 46832 the files. Between the time a pathname is created and the file is opened, it is possible for some
 46833 other process to create a file with the same name. Applications may find *tmpfile()* more useful.

46834 **RATIONALE**

46835 None.

46836 **FUTURE DIRECTIONS**

46837 None.

46838 **SEE ALSO**

46839 *fopen()*, *free()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink()*, the Base Definitions volume of
46840 IEEE Std 1003.1-2001, <**stdio.h**>

46841 **CHANGE HISTORY**

46842 First released in Issue 1. Derived from Issue 1 of the SVID.

46843 **Issue 5**

46844 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
46845 previous issues.

46846 **NAME**

46847 tfind — search binary search tree

46848 **SYNOPSIS**

46849 XSI #include <search.h>

46850 void *tfind(const void *key, void *const *rootp,
46851 int (*compar)(const void *, const void *));

46852

46853 **DESCRIPTION**46854 Refer to *tdelete()*.

46855 **NAME**

46856 tgamma, tgammaf, tgammal — compute gamma() function

46857 **SYNOPSIS**

46858 #include <math.h>

46859 double tgamma(double x);

46860 float tgammaf(float x);

46861 long double tgammal(long double x);

46862 **DESCRIPTION**

46863 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 46864 conflict between the requirements described here and the ISO C standard is unintentional. This
 46865 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

46866 These functions shall compute the *gamma()* function of *x*.

46867 An application wishing to check for error situations should set *errno* to zero and call
 46868 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 46869 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 46870 zero, an error has occurred.

46871 **RETURN VALUE**46872 Upon successful completion, these functions shall return *Gamma(x)*.

46873 If *x* is a negative integer, a domain error shall occur, and either a NaN (if supported), or an
 46874 implementation-defined value shall be returned.

46875 If the correct value would cause overflow, a range error shall occur and *tgamma()*, *tgammaf()*,
 46876 and *tgammal()* shall return $\pm\text{HUGE\_VAL}$, $\pm\text{HUGE\_VALF}$, or $\pm\text{HUGE\_VALL}$, respectively, with
 46877 the same sign as the correct value of the function.

46878 MX If *x* is NaN, a NaN shall be returned.46879 If *x* is +Inf, *x* shall be returned.

46880 If *x* is ± 0 , a pole error shall occur, and *tgamma()*, *tgammaf()*, and *tgammal()* shall return
 46881 $\pm\text{HUGE\_VAL}$, $\pm\text{HUGE\_VALF}$, and $\pm\text{HUGE\_VALL}$, respectively.

46882 If *x* is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-
 46883 defined value shall be returned.

46884 **ERRORS**

46885 These functions shall fail if:

46886 MX Domain Error The value of *x* is a negative integer, or *x* is -Inf.

46887 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 46888 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 46889 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 46890 shall be raised.

46891 MX Pole Error The value of *x* is zero.

46892 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 46893 then *errno* shall be set to [ERANGE]. If the integer expression
 46894 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
 46895 zero floating-point exception shall be raised.

46896 Range Error The value overflows.

46897 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,
 46898 then *errno* shall be set to [ERANGE]. If the integer expression
 46899 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the overflow
 46900 floating-point exception shall be raised.

46901 **EXAMPLES**

46902 None.

46903 **APPLICATION USAGE**

46904 For IEEE Std 754-1985 **double**, overflow happens when $0 < x < 1/\text{DBL\_MAX}$, and $171.7 < x$.

46905 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
 46906 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

46907 **RATIONALE**

46908 This function is named *tgamma()* in order to avoid conflicts with the historical *gamma()* and
 46909 *lgamma()* functions.

46910 **FUTURE DIRECTIONS**

46911 It is possible that the error response for a negative integer argument may be changed to a pole
 46912 error and a return value of $\pm\text{Inf}$.

46913 **SEE ALSO**

46914 *feclearexcept()*, *fetestexcept()*, *lgamma()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 46915 Section 4.18, Treatment of Error Conditions for Mathematical Functions, **<math.h>**

46916 **CHANGE HISTORY**

46917 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

46918 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/65 is applied, correcting the third |
 46919 paragraph in the RETURN VALUE section. |

46920 **NAME**

46921 time — get time

46922 **SYNOPSIS**

46923 #include <time.h>

46924 time_t time(time_t *tloc);

46925 **DESCRIPTION**

46926 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 46927 conflict between the requirements described here and the ISO C standard is unintentional. This
 46928 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

46929 CX The *time()* function shall return the value of time in seconds since the Epoch.

46930 The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer,
 46931 no value is stored.

46932 **RETURN VALUE**

46933 Upon successful completion, *time()* shall return the value of time. Otherwise, (**time_t**)−1 shall be
 46934 returned.

46935 **ERRORS**

46936 No errors are defined.

46937 **EXAMPLES**46938 **Getting the Current Time**

46939 The following example uses the *time()* function to calculate the time elapsed, in seconds, since
 46940 the Epoch, *localtime()* to convert that value to a broken-down time, and *asctime()* to convert the
 46941 broken-down time values into a printable string.

```
46942       #include <stdio.h>
46943       #include <time.h>

46944       int main(void)
46945       {
46946       time_t result;

46947           result = time(NULL);
46948           printf("%s%ju secs since the Epoch\n",
46949                asctime(localtime(&result)),
46950                (uintmax_t)result);
46951           return(0);
46952       }
```

46953 This example writes the current time to *stdout* in a form like this:

```
46954       Wed Jun 26 10:32:15 1996
46955       835810335 secs since the Epoch
```

Timing an Event

The following example gets the current time, prints it out in the user's format, and prints the number of minutes to an event being timed.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
minutes_to_event = ...;
printf("The time is ");
puts(asctime(localtime(&now)));
printf("There are %d minutes to the event.\n",
      minutes_to_event);
...
```

APPLICATION USAGE

None.

RATIONALE

The *time()* function returns a value in seconds (type **time_t**) while *times()* returns a set of values in clock ticks (type **clock_t**). Some historical implementations, such as 4.3 BSD, have mechanisms capable of returning more precise times (see below). A generalized timing scheme to unify these various timing mechanisms has been proposed but not adopted.

Implementations in which **time_t** is a 32-bit signed integer (many historical implementations) fail in the year 2038. IEEE Std 1003.1-2001 does not address this problem. However, the use of the **time_t** type is mandated in order to ease the eventual fix.

The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C standard.

Many historical implementations (including Version 7) and the 1984 /usr/group standard use **long** instead of **time_t**. This volume of IEEE Std 1003.1-2001 uses the latter type in order to agree with the ISO C standard.

4.3 BSD includes *time()* only as an alternate function to the more flexible *gettimeofday()* function.

FUTURE DIRECTIONS

In a future version of this volume of IEEE Std 1003.1-2001, **time_t** is likely to be required to be capable of representing times far in the future. Whether this will be mandated as a 64-bit type or a requirement that a specific date in the future be representable (for example, 10000 AD) is not yet determined. Systems purchased after the approval of this volume of IEEE Std 1003.1-2001 should be evaluated to determine whether their lifetime will extend past 2038.

SEE ALSO

asctime(), *clock()*, *ctime()*, *difftime()*, *gettimeofday()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *utime()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

46999 **Issue 6**

47000 Extensions beyond the ISO C standard are marked.

47001 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

47002 **NAME**47003 timer_create — create a per-process timer (**REALTIME**)47004 **SYNOPSIS**

```
47005 TMR    #include <signal.h>
47006         #include <time.h>
```

```
47007         int timer_create(clockid_t clockid, struct sigevent *restrict evp,
47008                         timer_t *restrict timerid);
47009
```

47010 **DESCRIPTION**

47011 The *timer_create()* function shall create a per-process timer using the specified clock, *clock_id*, as
 47012 the timing base. The *timer_create()* function shall return, in the location referenced by *timerid*, a
 47013 timer ID of type **timer_t** used to identify the timer in timer requests. This timer ID shall be
 47014 unique within the calling process until the timer is deleted. The particular clock, *clock_id*, is
 47015 defined in **<time.h>**. The timer whose ID is returned shall be in a disarmed state upon return
 47016 from *timer_create()*.

47017 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the
 47018 application, defines the asynchronous notification to occur as specified in Section 2.4.1 (on page
 47019 28) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument
 47020 pointed to a **sigevent** structure with the *sigev_notify* member having the value SIGEV_SIGNAL,
 47021 the *sigev_signo* having a default signal number, and the *sigev_value* member having the value of
 47022 the timer ID.

47023 Each implementation shall define a set of clocks that can be used as timing bases for per-process
 47024 MON timers. All implementations shall support a *clock_id* of CLOCK_REALTIME. If the Monotonic
 47025 Clock option is supported, implementations shall support a *clock_id* of CLOCK_MONOTONIC.

47026 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed
 47027 and deleted by an *exec*.

47028 CPT If _POSIX_CPUTIME is defined, implementations shall support *clock_id* values representing the
 47029 CPU-time clock of the calling process.

47030 TCT If _POSIX_THREAD_CPUTIME is defined, implementations shall support *clock_id* values
 47031 representing the CPU-time clock of the calling thread.

47032 CPT|TCT It is implementation-defined whether a *timer_create()* function will succeed if the value defined
 47033 by *clock_id* corresponds to the CPU-time clock of a process or thread different from the process
 47034 or thread invoking the function.

47035 **RETURN VALUE**

47036 If the call succeeds, *timer_create()* shall return zero and update the location referenced by *timerid*
 47037 to a **timer_t**, which can be passed to the per-process timer calls. If an error occurs, the function
 47038 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if
 47039 an error occurs.

47040 **ERRORS**

47041 The *timer_create()* function shall fail if:

47042	[EAGAIN]	The system lacks sufficient signal queuing resources to honor the request.
47043	[EAGAIN]	The calling process has already created all of the timers it is allowed by this
47044		implementation.
47045	[EINVAL]	The specified clock ID is not defined.

47046 CPT|TCT [ENOTSUP] The implementation does not support the creation of a timer attached to the
 47047 CPU-time clock that is specified by *clock_id* and associated with a process or
 47048 thread different from the process or thread invoking *timer_create()*.

47049 EXAMPLES

47050 None.

47051 APPLICATION USAGE

47052 None.

47053 RATIONALE

47054 Periodic Timer Overrun and Resource Allocation

47055 The specified timer facilities may deliver realtime signals (that is, queued signals) on
 47056 implementations that support this option. Since realtime applications cannot afford to lose
 47057 notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it
 47058 must be possible to ensure that sufficient resources exist to deliver the signal when the event
 47059 occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a
 47060 request and a subsequent signal generation. If the request cannot allocate the signal delivery
 47061 resources, it can fail the call with an [EAGAIN] error.

47062 Periodic timers are a special case. A single request can generate an unspecified number of
 47063 signals. This is not a problem if the requesting process can service the signals as fast as they are
 47064 generated, thus making the signal delivery resources available for delivery of subsequent
 47065 periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic
 47066 timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the
 47067 currently pending signal has been delivered.

47068 Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a
 47069 pending signal are generated, it is implementation-defined whether a signal is delivered for each
 47070 occurrence. This is not adequate for some realtime applications. So a mechanism is required to
 47071 allow applications to detect how many timer expirations were delayed without requiring an
 47072 indefinite amount of system resources to store the delayed expirations.

47073 The specified facilities provide for an overrun count. The overrun count is defined as the number
 47074 of extra timer expirations that occurred between the time a timer expiration signal is generated
 47075 and the time the signal is delivered. The signal-catching function, if it is concerned with
 47076 overruns, can retrieve this count on entry. With this method, a periodic timer only needs one
 47077 “signal queuing resource” that can be allocated at the time of the *timer_create()* function call.

47078 A function is defined to retrieve the overrun count so that an application need not allocate static
 47079 storage to contain the count, and an implementation need not update this storage
 47080 asynchronously on timer expirations. But, for some high-frequency periodic applications, the
 47081 overhead of an additional system call on each timer expiration may be prohibitive. The
 47082 functions, as defined, permit an implementation to maintain the overrun count in user space,
 47083 associated with the *timerid*. The *timer_getoverrun()* function can then be implemented as a macro
 47084 that uses the *timerid* argument (which may just be a pointer to a user space structure containing
 47085 the counter) to locate the overrun count with no system call overhead. Other implementations,
 47086 less concerned with this class of applications, can avoid the asynchronous update of user space
 47087 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

47088 **Timer Expiration Signal Parameters**

47089 The Realtime Signals Extension option supports an application-specific datum that is delivered
47090 to the extended signal handler. This value is explicitly specified by the application, along with
47091 the signal number to be delivered, in a **sigevent** structure. The type of the application-defined
47092 value can be either an integer constant or a pointer. This explicit specification of the value, as
47093 opposed to always sending the timer ID, was selected based on existing practice.

47094 It is common practice for realtime applications (on non-POSIX systems or realtime extended
47095 POSIX systems) to use the parameters of event handlers as the case label of a switch statement
47096 or as a pointer to an application-defined data structure. Since *timer_ids* are dynamically allocated
47097 by the *timer_create()* function, they can be used for neither of these functions without additional
47098 application overhead in the signal handler; for example, to search an array of saved timer IDs to
47099 associate the ID with a constant or application data structure.

47100 **FUTURE DIRECTIONS**

47101 None.

47102 **SEE ALSO**

47103 *clock_getres()*, *timer_delete()*, *timer_getoverrun()*, the Base Definitions volume of
47104 IEEE Std 1003.1-2001, <**time.h**>

47105 **CHANGE HISTORY**

47106 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47107 **Issue 6**

47108 The *timer_create()* function is marked as part of the Timers option.

47109 The [ENOSYS] error condition has been removed as stubs need not be provided if an
47110 implementation does not support the Timers option.

47111 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

47112 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the
47113 requirement for the CLOCK_MONOTONIC clock under the Monotonic Clock option.

47114 The **restrict** keyword is added to the *timer_create()* prototype for alignment with the
47115 ISO/IEC 9899:1999 standard.

47116 **NAME**

47117 timer_delete — delete a per-process timer (**REALTIME**)

47118 **SYNOPSIS**

47119 TMR `#include <time.h>`

47120 `int timer_delete(timer_t timerid);`

47121

47122 **DESCRIPTION**

47123 The *timer_delete()* function deletes the specified timer, *timerid*, previously created by the
47124 *timer_create()* function. If the timer is armed when *timer_delete()* is called, the behavior shall be
47125 as if the timer is automatically disarmed before removal. The disposition of pending signals for
47126 the deleted timer is unspecified.

47127 **RETURN VALUE**

47128 If successful, the *timer_delete()* function shall return a value of zero. Otherwise, the function shall
47129 return a value of `-1` and set *errno* to indicate the error.

47130 **ERRORS**

47131 The *timer_delete()* function shall fail if:

47132 [EINVAL] The timer ID specified by *timerid* is not a valid timer ID.

47133 **EXAMPLES**

47134 None.

47135 **APPLICATION USAGE**

47136 None.

47137 **RATIONALE**

47138 None.

47139 **FUTURE DIRECTIONS**

47140 None.

47141 **SEE ALSO**

47142 *timer_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

47143 **CHANGE HISTORY**

47144 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47145 **Issue 6**

47146 The *timer_delete()* function is marked as part of the Timers option.

47147 The [ENOSYS] error condition has been removed as stubs need not be provided if an
47148 implementation does not support the Timers option.

47149 **NAME**47150 timer_getoverrun, timer_gettime, timer_settime — per-process timers (**REALTIME**)47151 **SYNOPSIS**47152 TMR `#include <time.h>`

```

47153 int timer_getoverrun(timer_t timerid);
47154 int timer_gettime(timer_t timerid, struct itimerspec *value);
47155 int timer_settime(timer_t timerid, int flags,
47156                  const struct itimerspec *restrict value,
47157                  struct itimerspec *restrict ovalue);
47158

```

47159 **DESCRIPTION**

47160 The *timer_gettime()* function shall store the amount of time until the specified timer, *timerid*,
 47161 expires and the reload value of the timer into the space pointed to by the *value* argument. The
 47162 *it_value* member of this structure shall contain the amount of time before the timer expires, or
 47163 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if
 47164 the timer was armed with absolute time. The *it_interval* member of *value* shall contain the reload
 47165 value last set by *timer_settime()*.

47166 The *timer_settime()* function shall set the time until the next expiration of the timer specified by
 47167 *timerid* from the *it_value* member of the *value* argument and arm the timer if the *it_value* member
 47168 of *value* is non-zero. If the specified timer was already armed when *timer_settime()* is called, this
 47169 call shall reset the time until next expiration to the *value* specified. If the *it_value* member of *value*
 47170 is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending
 47171 expiration notifications is unspecified.

47172 If the flag **TIMER_ABSTIME** is not set in the argument *flags*, *timer_settime()* shall behave as if the
 47173 time until next expiration is set to be equal to the interval specified by the *it_value* member of
 47174 *value*. That is, the timer shall expire in *it_value* nanoseconds from when the call is made. If the
 47175 flag **TIMER_ABSTIME** is set in the argument *flags*, *timer_settime()* shall behave as if the time
 47176 until next expiration is set to be equal to the difference between the absolute time specified by
 47177 the *it_value* member of *value* and the current value of the clock associated with *timerid*. That is,
 47178 the timer shall expire when the clock reaches the value specified by the *it_value* member of *value*.
 47179 If the specified time has already passed, the function shall succeed and the expiration
 47180 notification shall be made.

47181 The reload value of the timer shall be set to the value specified by the *it_interval* member of
 47182 *value*. When a timer is armed with a non-zero *it_interval*, a periodic (or repetitive) timer is
 47183 specified.

47184 Time values that are between two consecutive non-negative integer multiples of the resolution
 47185 of the specified timer shall be rounded up to the larger multiple of the resolution. Quantization
 47186 error shall not cause the timer to expire earlier than the rounded time value.

47187 If the argument *ovalue* is not NULL, the *timer_settime()* function shall store, in the location
 47188 referenced by *ovalue*, a value representing the previous amount of time before the timer would
 47189 have expired, or zero if the timer was disarmed, together with the previous timer reload value.
 47190 Timers shall not expire before their scheduled time.

47191 Only a single signal shall be queued to the process for a given timer at any point in time. When a
 47192 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun
 47193 shall occur. When a timer expiration signal is delivered to or accepted by a process, if the
 47194 implementation supports the Realtime Signals Extension, the *timer_getoverrun()* function shall
 47195 return the timer expiration overrun count for the specified timer. The overrun count returned
 47196 contains the number of extra timer expirations that occurred between the time the signal was

generated (queued) and when it was delivered or accepted, up to but not including an implementation-defined maximum of {DELAYTIMER_MAX}. If the number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the overrun count shall be set to {DELAYTIMER_MAX}. The value returned by *timer_getoverrun()* shall apply to the most recent expiration signal delivery or acceptance for the timer. If no expiration signal has been delivered for the timer, or if the Realtime Signals Extension is not supported, the return value of *timer_getoverrun()* is unspecified.

47204 RETURN VALUE

47205 If the *timer_getoverrun()* function succeeds, it shall return the timer expiration overrun count as
47206 explained above.

47207 If the *timer_gettime()* or *timer_settime()* functions succeed, a value of 0 shall be returned.

47208 If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to
47209 indicate the error.

47210 ERRORS

47211 The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions shall fail if:

47212 [EINVAL] The *timerid* argument does not correspond to an ID returned by *timer_create()*
47213 but not yet deleted by *timer_delete()*.

47214 The *timer_settime()* function shall fail if:

47215 [EINVAL] A *value* structure specified a nanosecond value less than zero or greater than
47216 or equal to 1 000 million, and the *it_value* member of that structure did not
47217 specify zero seconds and nanoseconds.

47218 EXAMPLES

47219 None.

47220 APPLICATION USAGE

47221 None.

47222 RATIONALE

47223 Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The
47224 inverse of this tick rate is the clock resolution, also called the clock granularity, which in either
47225 case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for
47226 these common rates. The granularity of practical clocks implies that if one reads a given clock
47227 twice in rapid succession, one may get the same time value twice; and that timers must wait for
47228 the next clock tick after the theoretical expiration time, to ensure that a timer never returns too
47229 soon. Note also that the granularity of the clock may be significantly coarser than the resolution
47230 of the data format used to set and get time and interval values. Also note that some
47231 implementations may choose to adjust time and/or interval values to exactly match the ticks of
47232 the underlying clock.

47233 This volume of IEEE Std 1003.1-2001 defines functions that allow an application to determine the
47234 implementation-supported resolution for the clocks and requires an implementation to
47235 document the resolution supported for timers and *nanosleep()* if they differ from the supported
47236 clock resolution. This is more of a procurement issue than a runtime application issue.

47237 FUTURE DIRECTIONS

47238 None.

47239 **SEE ALSO**

47240 *clock_getres()*, *timer_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

47241 **CHANGE HISTORY**

47242 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47243 **Issue 6**

47244 The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions are marked as part of the
47245 Timers option.

47246 The [ENOSYS] error condition has been removed as stubs need not be provided if an
47247 implementation does not support the Timers option.

47248 The [EINVAL] error condition is updated to include the following: “and the *it_value* member of
47249 that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC
47250 Interpretation 1003.1 #89.

47251 The DESCRIPTION for *timer_getoverrun()* is updated to clarify that “If no expiration signal has
47252 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return
47253 value of *timer_getoverrun()* is unspecified”.

47254 The **restrict** keyword is added to the *timer_settime()* prototype for alignment with the
47255 ISO / IEC 9899:1999 standard.

47256 **NAME**

47257 times — get process and waited-for child process times

47258 **SYNOPSIS**

47259 #include <sys/times.h>

47260 clock_t times(struct tms *buffer);

47261 **DESCRIPTION**47262 The *times()* function shall fill the **tms** structure pointed to by *buffer* with time-accounting
47263 information. The **tms** structure is defined in <sys/times.h>.

47264 All times are measured in terms of the number of clock ticks used.

47265 The times of a terminated child process shall be included in the *tms_cutime* and *tms_cstime*
47266 elements of the parent when *wait()* or *waitpid()* returns the process ID of this terminated child. If
47267 a child process has not waited for its children, their times shall not be included in its times.47268 • The *tms_utime* structure member is the CPU time charged for the execution of user
47269 instructions of the calling process.47270 • The *tms_stime* structure member is the CPU time charged for execution by the system on
47271 behalf of the calling process.47272 • The *tms_cutime* structure member is the sum of the *tms_utime* and *tms_cutime* times of the
47273 child processes.47274 • The *tms_cstime* structure member is the sum of the *tms_stime* and *tms_cstime* times of the child
47275 processes.47276 **RETURN VALUE**47277 Upon successful completion, *times()* shall return the elapsed real time, in clock ticks, since an
47278 arbitrary point in the past (for example, system start-up time). This point does not change from
47279 one invocation of *times()* within the process to another. The return value may overflow the
47280 possible range of type **clock_t**. If *times()* fails, (**clock_t**)-1 shall be returned and *errno* set to
47281 indicate the error.47282 **ERRORS**

47283 No errors are defined.

47284 **EXAMPLES**47285 **Timing a Database Lookup**47286 The following example defines two functions, *start_clock()* and *end_clock()*, that are used to time
47287 a lookup. It also defines variables of type **clock_t** and **tms** to measure the duration of
47288 transactions. The *start_clock()* function saves the beginning times given by the *times()* function.
47289 The *end_clock()* function gets the ending times and prints the difference between the two times.47290 #include <sys/times.h>
47291 #include <stdio.h>
47292 ...
47293 void start_clock(void);
47294 void end_clock(char *msg);
47295 ...
47296 static clock_t st_time;
47297 static clock_t en_time;
47298 static struct tms st_cpu;
47299 static struct tms en_cpu;

```

47300     ...
47301     void
47302     start_clock()
47303     {
47304         st_time = times(&st_cpu);
47305     }

47306     /* This example assumes that the result of each subtraction
47307        is within the range of values that can be represented in
47308        an integer type. */
47309     void
47310     end_clock(char *msg)
47311     {
47312         en_time = times(&en_cpu);

47313         fputs(msg, stdout);
47314         printf("Real Time: %jd, User Time %jd, System Time %jd\n",
47315             (intmax_t)(en_time - st_time),
47316             (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
47317             (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
47318     }

```

47319 APPLICATION USAGE

47320 Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per
 47321 second as it may vary from system to system.

47322 RATIONALE

47323 The accuracy of the times reported is intentionally left unspecified to allow implementations
 47324 flexibility in design, from uniprocessor to multi-processor networks.

47325 The inclusion of times of child processes is recursive, so that a parent process may collect the
 47326 total times of all of its descendants. But the times of a child are only added to those of its parent
 47327 when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process
 47328 can always see the total times of all its descendants; see also the discussion of the term
 47329 “realtime” in *alarm()*.

47330 If the type **clock_t** is defined to be a signed 32-bit integer, it overflows in somewhat more than a
 47331 year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual
 47332 systems that run continuously for longer than that. This volume of IEEE Std 1003.1-2001 permits
 47333 an implementation to make the reference point for the returned value be the start-up time of the
 47334 process, rather than system start-up time.

47335 The term “charge” in this context has nothing to do with billing for services. The operating
 47336 system accounts for time used in this way. That information must be correct, regardless of how
 47337 that information is used.

47338 FUTURE DIRECTIONS

47339 None.

47340 SEE ALSO

47341 *alarm()*, *exec*, *fork()*, *sysconf()*, *time()*, *wait()*, the Base Definitions volume of
 47342 IEEE Std 1003.1-2001, <**sys/times.h**>

47343 CHANGE HISTORY

47344 First released in Issue 1. Derived from Issue 1 of the SVID.

47345 NAME

47346 timezone — difference from UTC and local standard time

47347 SYNOPSIS

47348 xSI #include <time.h>

47349 extern long timezone;

47350

47351 DESCRIPTION

47352 Refer to *tzset()*.

47353 **NAME**

47354 tmpfile — create a temporary file

47355 **SYNOPSIS**

47356 #include <stdio.h>

47357 FILE *tmpfile(void);

47358 **DESCRIPTION**

47359 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 47360 conflict between the requirements described here and the ISO C standard is unintentional. This
 47361 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47362 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file
 47363 shall be automatically deleted when all references to the file are closed. The file is opened as in
 47364 *fopen()* for update (w+).

47365 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is
 47366 killed while it is processing a call to *tmpfile()*.

47367 An error message may be written to standard error if the stream cannot be opened.

47368 **RETURN VALUE**

47369 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is
 47370 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

47371 **ERRORS**47372 The *tmpfile()* function shall fail if:

47373 CX [EINTR] A signal was caught during *tmpfile()*.

47374 CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

47375 CX [ENFILE] The maximum allowable number of files is currently open in the system.

47376 CX [ENOSPC] The directory or file system which would contain the new file cannot be
 47377 expanded.

47378 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly
 47379 in an object of type **off_t**.

47380 The *tmpfile()* function may fail if:

47381 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

47382 CX [ENOMEM] Insufficient storage space is available.

47383 **EXAMPLES**47384 **Creating a Temporary File**

47385 The following example creates a temporary file for update, and returns a pointer to a stream for
 47386 the created file in the *fp* variable.

47387 #include <stdio.h>

47388 ...

47389 FILE *fp;

47390 fp = tmpfile ();

47391 **APPLICATION USAGE**

47392 It should be possible to open at least {TMP_MAX} temporary files during the lifetime of the
47393 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number
47394 simultaneously open other than this limit and any limit on the number of open files
47395 ({FOPEN_MAX}).

47396 **RATIONALE**

47397 None.

47398 **FUTURE DIRECTIONS**

47399 None.

47400 **SEE ALSO**

47401 *fopen()*, *tmpnam()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>

47402 **CHANGE HISTORY**

47403 First released in Issue 1. Derived from Issue 1 of the SVID.

47404 **Issue 5**

47405 Large File Summit extensions are added.

47406 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
47407 in previous issues.

47408 **Issue 6**

47409 Extensions beyond the ISO C standard are marked.

47410 The following new requirements on POSIX implementations derive from alignment with the
47411 Single UNIX Specification:

- 47412 • In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support
47413 large files.
- 47414 • The [EMFILE] optional error condition is added.

47415 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999
47416 standard.

47417 **NAME**

47418 tmpnam — create a name for a temporary file

47419 **SYNOPSIS**

47420 #include <stdio.h>

47421 char *tmpnam(char *s);

47422 **DESCRIPTION**

47423 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 47424 conflict between the requirements described here and the ISO C standard is unintentional. This
 47425 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47426 The *tmpnam()* function shall generate a string that is a valid filename and that is not the same as
 47427 the name of an existing file. The function is potentially capable of generating {TMP_MAX}
 47428 different strings, but any or all of them may already be in use by existing files and thus not be
 47429 suitable return values.

47430 The *tmpnam()* function generates a different string each time it is called from the same process,
 47431 up to {TMP_MAX} times. If it is called more than {TMP_MAX} times, the behavior is
 47432 implementation-defined.

47433 The implementation shall behave as if no function defined in this volume of
 47434 IEEE Std 1003.1-2001 calls *tmpnam()*.

47435 CX If the application uses any of the functions guaranteed to be available if either
 47436 _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS is defined, the application shall
 47437 ensure that the *tmpnam()* function is called with a non-NULL parameter.

47438 **RETURN VALUE**

47439 Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can
 47440 be generated, the *tmpnam()* function shall return a null pointer.

47441 If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and
 47442 return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the
 47443 argument *s* is not a null pointer, it is presumed to point to an array of at least L_tmpnam **chars**;
 47444 *tmpnam()* shall write its result in that array and shall return the argument as its value.

47445 **ERRORS**

47446 No errors are defined.

47447 **EXAMPLES**47448 **Generating a Filename**47449 The following example generates a unique filename and stores it in the array pointed to by *ptr*.

47450 #include <stdio.h>

47451 ...

47452 char filename[L_tmpnam+1];

47453 char *ptr;

47454 ptr = tmpnam(filename);

47455 **APPLICATION USAGE**

47456 This function only creates filenames. It is the application's responsibility to create and remove
 47457 the files.

47458 Between the time a pathname is created and the file is opened, it is possible for some other
 47459 process to create a file with the same name. Applications may find *tmpfile()* more useful.

47460 **RATIONALE**

47461 None.

47462 **FUTURE DIRECTIONS**

47463 None.

47464 **SEE ALSO**

47465 *fopen()*, *open()*, *tempnam()*, *tmpfile()*, *unlink()*, the Base Definitions volume of
47466 IEEE Std 1003.1-2001, **<stdio.h>**

47467 **CHANGE HISTORY**

47468 First released in Issue 1. Derived from Issue 1 of the SVID.

47469 **Issue 5**

47470 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

47471 **Issue 6**

47472 Extensions beyond the ISO C standard are marked.

47473 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47474 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

47475 **NAME**

47476 toascii — translate an integer to a 7-bit ASCII character

47477 **SYNOPSIS**

47478 xSI #include <ctype.h>

47479 int toascii(int c);

47480

47481 **DESCRIPTION**47482 The *toascii()* function shall convert its argument into a 7-bit ASCII character.47483 **RETURN VALUE**47484 The *toascii()* function shall return the value (*c* & 0x7f).47485 **ERRORS**

47486 No errors are returned.

47487 **EXAMPLES**

47488 None.

47489 **APPLICATION USAGE**

47490 None.

47491 **RATIONALE**

47492 None.

47493 **FUTURE DIRECTIONS**

47494 None.

47495 **SEE ALSO**47496 *isascii()*, the Base Definitions volume of IEEE Std 1003.1-2001, <ctype.h>47497 **CHANGE HISTORY**

47498 First released in Issue 1. Derived from Issue 1 of the SVID.

47499 NAME

47500 tolower — transliterate uppercase characters to lowercase

47501 SYNOPSIS

47502 #include <ctype.h>

47503 int tolower(int c);

47504 DESCRIPTION

47505 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47506 conflict between the requirements described here and the ISO C standard is unintentional. This
47507 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47508 The *tolower()* function has as a domain a type **int**, the value of which is representable as an
47509 **unsigned char** or the value of EOF. If the argument has any other value, the behavior is
47510 undefined. If the argument of *tolower()* represents an uppercase letter, and there exists a
47511 CX corresponding lowercase letter (as defined by character type information in the program locale
47512 category *LC_CTYPE*), the result shall be the corresponding lowercase letter. All other arguments
47513 in the domain are returned unchanged.

47514 RETURN VALUE

47515 Upon successful completion, *tolower()* shall return the lowercase letter corresponding to the
47516 argument passed; otherwise, it shall return the argument unchanged.

47517 ERRORS

47518 No errors are defined.

47519 EXAMPLES

47520 None.

47521 APPLICATION USAGE

47522 None.

47523 RATIONALE

47524 None.

47525 FUTURE DIRECTIONS

47526 None.

47527 SEE ALSO

47528 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, **<ctype.h>**

47529 CHANGE HISTORY

47530 First released in Issue 1. Derived from Issue 1 of the SVID.

47531 Issue 6

47532 Extensions beyond the ISO C standard are marked.

47533 **NAME**

47534 toupper — transliterate lowercase characters to uppercase

47535 **SYNOPSIS**

47536 #include <ctype.h>

47537 int toupper(int c);

47538 **DESCRIPTION**

47539 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47540 conflict between the requirements described here and the ISO C standard is unintentional. This
47541 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47542 The *toupper()* function has as a domain a type **int**, the value of which is representable as an
47543 **unsigned char** or the value of EOF. If the argument has any other value, the behavior is
47544 undefined. If the argument of *toupper()* represents a lowercase letter, and there exists a
47545 CX corresponding uppercase letter (as defined by character type information in the program locale
47546 category *LC_CTYPE*), the result shall be the corresponding uppercase letter. All other arguments
47547 in the domain are returned unchanged.

47548 **RETURN VALUE**

47549 Upon successful completion, *toupper()* shall return the uppercase letter corresponding to the
47550 argument passed.

47551 **ERRORS**

47552 No errors are defined.

47553 **EXAMPLES**

47554 None.

47555 **APPLICATION USAGE**

47556 None.

47557 **RATIONALE**

47558 None.

47559 **FUTURE DIRECTIONS**

47560 None.

47561 **SEE ALSO**47562 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>47563 **CHANGE HISTORY**

47564 First released in Issue 1. Derived from Issue 1 of the SVID.

47565 **Issue 6**

47566 Extensions beyond the ISO C standard are marked.

47567 **NAME**

47568 towctrans — wide-character transliteration

47569 **SYNOPSIS**

47570 #include <wctype.h>

47571 wint_t towctrans(wint_t *wc*, wctrans_t *desc*);47572 **DESCRIPTION**

47573 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 47574 conflict between the requirements described here and the ISO C standard is unintentional. This
 47575 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47576 The *towctrans()* function shall transliterate the wide-character code *wc* using the mapping
 47577 described by *desc*. The current setting of the *LC_CTYPE* category should be the same as during
 47578 CX the call to *wctrans()* that returned the value *desc*. If the value of *desc* is invalid (that is, not
 47579 obtained by a call to *wctrans()* or *desc* is invalidated by a subsequent call to *setlocale()* that has
 47580 affected category *LC_CTYPE*), the result is unspecified.

47581 An application wishing to check for error situations should set *errno* to 0 before calling
 47582 *towctrans()*. If *errno* is non-zero on return, an error has occurred.

47583 **RETURN VALUE**

47584 If successful, the *towctrans()* function shall return the mapped value of *wc* using the mapping
 47585 described by *desc*. Otherwise, it shall return *wc* unchanged.

47586 **ERRORS**47587 The *towctrans()* function may fail if:

47588 CX [EINVAL] *desc* contains an invalid transliteration descriptor.

47589 **EXAMPLES**

47590 None.

47591 **APPLICATION USAGE**

47592 The strings "tolower" and "toupper" are reserved for the standard mapping names. In the
 47593 table below, the functions in the left column are equivalent to the functions in the right column.

47594 tolower(<i>wc</i>)	towctrans(<i>wc</i> , wctrans("tolower"))
47595 toupper(<i>wc</i>)	towctrans(<i>wc</i> , wctrans("toupper"))

47596 **RATIONALE**

47597 None.

47598 **FUTURE DIRECTIONS**

47599 None.

47600 **SEE ALSO**

47601 *tolower()*, *toupper()*, *wctrans()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 47602 <wctype.h>

47603 **CHANGE HISTORY**

47604 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

47605 **Issue 6**

47606 Extensions beyond the ISO C standard are marked.

47607 **NAME**

47608 tolower — transliterate uppercase wide-character code to lowercase

47609 **SYNOPSIS**

47610 #include <wctype.h>

47611 wint_t tolower(wint_t wc);

47612 **DESCRIPTION**

47613 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 47614 conflict between the requirements described here and the ISO C standard is unintentional. This
 47615 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47616 The *tolower()* function has as a domain a type **wint_t**, the value of which the application shall
 47617 ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a
 47618 valid character in the current locale or the value of WEOF. If the argument has any other value,
 47619 the behavior is undefined. If the argument of *tolower()* represents an uppercase wide-character
 47620 code, and there exists a corresponding lowercase wide-character code (as defined by character
 47621 type information in the program locale category *LC_CTYPE*), the result shall be the
 47622 corresponding lowercase wide-character code. All other arguments in the domain are returned
 47623 unchanged.

47624 **RETURN VALUE**

47625 Upon successful completion, *tolower()* shall return the lowercase letter corresponding to the
 47626 argument passed; otherwise, it shall return the argument unchanged.

47627 **ERRORS**

47628 No errors are defined.

47629 **EXAMPLES**

47630 None.

47631 **APPLICATION USAGE**

47632 None.

47633 **RATIONALE**

47634 None.

47635 **FUTURE DIRECTIONS**

47636 None.

47637 **SEE ALSO**

47638 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <**wctype.h**>,
 47639 <**wchar.h**>

47640 **CHANGE HISTORY**

47641 First released in Issue 4.

47642 **Issue 5**

47643 The following change has been made in this issue for alignment with
 47644 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 47645 • The SYNOPSIS has been changed to indicate that this function and associated data types are
 47646 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

47647 **Issue 6**

47648 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47649 **NAME**

47650 towupper — transliterate lowercase wide-character code to uppercase

47651 **SYNOPSIS**

47652 #include <wctype.h>

47653 wint_t towupper(wint_t wc);

47654 **DESCRIPTION**

47655 cx The functionality described on this reference page is aligned with the ISO C standard. Any
47656 conflict between the requirements described here and the ISO C standard is unintentional. This
47657 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47658 The *towupper()* function has as a domain a type **wint_t**, the value of which the application shall
47659 ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a
47660 valid character in the current locale or the value of WEOF. If the argument has any other value,
47661 the behavior is undefined. If the argument of *towupper()* represents a lowercase wide-character
47662 code, and there exists a corresponding uppercase wide-character code (as defined by character
47663 type information in the program locale category *LC_CTYPE*), the result shall be the
47664 corresponding uppercase wide-character code. All other arguments in the domain are returned
47665 unchanged.

47666 **RETURN VALUE**

47667 Upon successful completion, *towupper()* shall return the uppercase letter corresponding to the
47668 argument passed. Otherwise, it shall return the argument unchanged.

47669 **ERRORS**

47670 No errors are defined.

47671 **EXAMPLES**

47672 None.

47673 **APPLICATION USAGE**

47674 None.

47675 **RATIONALE**

47676 None.

47677 **FUTURE DIRECTIONS**

47678 None.

47679 **SEE ALSO**

47680 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <**wctype.h**>,
47681 <**wchar.h**>

47682 **CHANGE HISTORY**

47683 First released in Issue 4.

47684 **Issue 5**

47685 The following change has been made in this issue for alignment with
47686 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 47687 • The SYNOPSIS has been changed to indicate that this function and associated data types are
47688 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

47689 **Issue 6**

47690 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47691 **NAME**

47692 trunc, truncf, trunc1 — round to truncated integer value

47693 **SYNOPSIS**

47694 #include <math.h>

47695 double trunc(double x);

47696 float truncf(float x);

47697 long double trunc1(long double x);

47698 **DESCRIPTION**47699 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47700 conflict between the requirements described here and the ISO C standard is unintentional. This
47701 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.47702 These functions shall round their argument to the integer value, in floating format, nearest to but
47703 no larger in magnitude than the argument.47704 **RETURN VALUE**

47705 Upon successful completion, these functions shall return the truncated integer value.

47706 MX If x is NaN, a NaN shall be returned.47707 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.47708 **ERRORS**

47709 No errors are defined.

47710 **EXAMPLES**

47711 None.

47712 **APPLICATION USAGE**

47713 None.

47714 **RATIONALE**

47715 None.

47716 **FUTURE DIRECTIONS**

47717 None.

47718 **SEE ALSO**

47719 The Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

47720 **CHANGE HISTORY**

47721 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

47722 **NAME**

47723 truncate — truncate a file to a specified length

47724 **SYNOPSIS**

47725 xSI #include <unistd.h>

47726 int truncate(const char *path, off_t length);

47727

47728 **DESCRIPTION**47729 The *truncate()* function shall cause the regular file named by *path* to have a size which shall be
47730 equal to *length* bytes.47731 If the file previously was larger than *length*, the extra data is discarded. If the file was previously
47732 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

47733 The application shall ensure that the process has write permission for the file.

47734 If the request would cause the file size to exceed the soft file size limit for the process, the
47735 request shall fail and the implementation shall generate the SIGXFSZ signal for the process.47736 This function shall not modify the file offset for any open file descriptions associated with the
47737 file. Upon successful completion, if the file size is changed, this function shall mark for update
47738 the *st_ctime* and *st_mtime* fields of the file, and the S_ISUID and S_ISGID bits of the file mode
47739 may be cleared.47740 **RETURN VALUE**47741 Upon successful completion, *truncate()* shall return 0. Otherwise, -1 shall be returned, and *errno*
47742 set to indicate the error.47743 **ERRORS**47744 The *truncate()* function shall fail if:

47745 [EINTR] A signal was caught during execution.

47746 [EINVAL] The *length* argument was less than 0.

47747 [EFBIG] or [EINVAL]

47748 The *length* argument was greater than the maximum file size.

47749 [EIO] An I/O error occurred while reading from or writing to a file system.

47750 [EACCES] A component of the path prefix denies search permission, or write permission
47751 is denied on the file.

47752 [EISDIR] The named file is a directory.

47753 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
47754 argument.

47755 [ENAMETOOLONG]

47756 The length of the *path* argument exceeds {PATH_MAX} or a pathname
47757 component is longer than {NAME_MAX}.47758 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.47759 [ENOTDIR] A component of the path prefix of *path* is not a directory.

47760 [EROFS] The named file resides on a read-only file system.

47761 The *truncate()* function may fail if:

47762 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
47763 resolution of the *path* argument.

47764 [ENAMETOOLONG]

47765 Pathname resolution of a symbolic link produced an intermediate result
47766 whose length exceeds {PATH_MAX}.

47767 **EXAMPLES**

47768 None.

47769 **APPLICATION USAGE**

47770 None.

47771 **RATIONALE**

47772 None.

47773 **FUTURE DIRECTIONS**

47774 None.

47775 **SEE ALSO**

47776 *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

47777 **CHANGE HISTORY**

47778 First released in Issue 4, Version 2.

47779 **Issue 5**

47780 Moved from X/OPEN UNIX extension to BASE.

47781 Large File Summit extensions are added.

47782 **Issue 6**

47783 This reference page is split out from the *ftruncate()* reference page.

47784 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47785 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
47786 [ELOOP] error condition is added.

47787 **NAME**

47788 truncf, trunc1 — round to truncated integer value

47789 **SYNOPSIS**

47790 #include <math.h>

47791 float truncf(float x);

47792 long double trunc1(long double x);

47793 **DESCRIPTION**47794 Refer to *trunc()*.

47795 **NAME**

47796 tsearch — search a binary search tree

47797 **SYNOPSIS**

47798 XSI #include <search.h>

47799 void *tsearch(const void *key, void **rootp,
47800 int (*compar)(const void *, const void *));

47801

47802 **DESCRIPTION**47803 Refer to *tdelete()*.

47804 NAME

47805 ttyname, ttyname_r — find the pathname of a terminal

47806 SYNOPSIS

47807 #include <unistd.h>

47808 char *ttyname(int *fildes*);

47809 TSF int ttyname_r(int *fildes*, char **name*, size_t *namesize*);

47810

47811 DESCRIPTION

47812 The *ttyname()* function shall return a pointer to a string containing a null-terminated pathname of the terminal associated with file descriptor *fildes*. The return value may point to static data whose content is overwritten by each call.

47815 The *ttyname()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

47817 TSF The *ttyname_r()* function shall store the null-terminated pathname of the terminal associated with the file descriptor *fildes* in the character array referenced by *name*. The array is *namesize* characters long and should have space for the name and the terminating null character. The maximum length of the terminal name shall be {TTY_NAME_MAX}.

47821 RETURN VALUE

47822 Upon successful completion, *ttyname()* shall return a pointer to a string. Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

47824 TSF If successful, the *ttyname_r()* function shall return zero. Otherwise, an error number shall be returned to indicate the error.

47826 ERRORS

47827 The *ttyname()* function may fail if:

47828 [EBADF] The *fildes* argument is not a valid file descriptor.

47829 [ENOTTY] The *fildes* argument does not refer to a terminal.

47830 The *ttyname_r()* function may fail if:

47831 TSF [EBADF] The *fildes* argument is not a valid file descriptor.

47832 TSF [ENOTTY] The *fildes* argument does not refer to a terminal.

47833 TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned including the terminating null character.

47835 EXAMPLES

47836 None.

47837 APPLICATION USAGE

47838 None.

47839 RATIONALE

47840 The term “terminal” is used instead of the historical term “terminal device” in order to avoid a reference to an undefined term.

47842 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

47845 **FUTURE DIRECTIONS**

47846 None.

47847 **SEE ALSO**

47848 The Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

47849 **CHANGE HISTORY**

47850 First released in Issue 1. Derived from Issue 1 of the SVID.

47851 **Issue 5**47852 The *ttyname_r()* function is included for alignment with the POSIX Threads Extension.

47853 A note indicating that the *ttyname()* function need not be reentrant is added to the
47854 DESCRIPTION.

47855 **Issue 6**47856 The *ttyname_r()* function is marked as part of the Thread-Safe Functions option.

47857 The following new requirements on POSIX implementations derive from alignment with the
47858 Single UNIX Specification:

- 47859 • The statement that *errno* is set on error is added.
- 47860 • The [EBADF] and [ENOTTY] optional error conditions are added.

47861 NAME

47862 twalk — traverse a binary search tree

47863 SYNOPSIS

47864 XSI `#include <search.h>`

47865 `void twalk(const void *root,`
47866 `void (*action)(const void *, VISIT, int ));`

47867

47868 DESCRIPTION

47869 Refer to *tdelete()*.

47870 NAME

47871 daylight, timezone, tzname, tzset — set timezone conversion information

47872 SYNOPSIS

47873 #include <time.h>

```
47874 xSI extern int daylight;
47875      extern long timezone;
47876 CX   extern char *tzname[2];
47877      void tzset(void);
```

47878

47879 DESCRIPTION

47880 The *tzset()* function shall use the value of the environment variable *TZ* to set time conversion information used by *ctime()*, *localtime()*, *mktime()*, and *strftime()*. If *TZ* is absent from the environment, implementation-defined default timezone information shall be used.

47883 The *tzset()* function shall set the external variable *tzname* as follows:

```
47884      tzname[0] = "std";
47885      tzname[1] = "dst";
```

47886 where *std* and *dst* are as described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

47888 xSI The *tzset()* function also shall set the external variable *daylight* to 0 if Daylight Savings Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable *timezone* shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

47892 RETURN VALUE

47893 The *tzset()* function shall not return a value.

47894 ERRORS

47895 No errors are defined.

47896 EXAMPLES

47897 Example *TZ* variables and their timezone differences are given in the table below:

	<i>TZ</i>	<i>timezone</i>
47900	EST5EDT	5*60*60
47901	GMT0	0*60*60
47902	JST-9	-9*60*60
47903	MET-1MEST	-1*60*60
47904	MST7MDT	7*60*60
47905	PST8PDT	8*60*60

47906 APPLICATION USAGE

47907 None.

47908 RATIONALE

47909 None.

47910 FUTURE DIRECTIONS

47911 None.

47912 **SEE ALSO**

47913 *ctime()*, *localtime()*, *mktime()*, *strftime()*, the Base Definitions volume of IEEE Std 1003.1-2001,
47914 Chapter 8, Environment Variables, **<time.h>**

47915 **CHANGE HISTORY**

47916 First released in Issue 1. Derived from Issue 1 of the SVID.

47917 **Issue 6**

47918 The example is corrected.

47919 **NAME**

47920 ualarm — set the interval timer

47921 **SYNOPSIS**

47922 OB XSI #include <unistd.h>

47923 useconds_t ualarm(useconds_t useconds, useconds_t interval);

47924

47925 **DESCRIPTION**

47926 The *ualarm()* function shall cause the SIGALRM signal to be generated for the calling process
 47927 after the number of realtime microseconds specified by the *useconds* argument has elapsed.
 47928 When the *interval* argument is non-zero, repeated timeout notification occurs with a period in
 47929 microseconds specified by the *interval* argument. If the notification signal, SIGALRM, is not
 47930 caught or ignored, the calling process is terminated.

47931 Implementations may place limitations on the granularity of timer values. For each interval
 47932 timer, if the requested timer value requires a finer granularity than the implementation supports,
 47933 the actual timer value shall be rounded up to the next supported value.

47934 Interactions between *ualarm()* and any of the following are unspecified:

47935 alarm()
 47936 nanosleep()
 47937 setitimer()
 47938 timer_create()
 47939 timer_delete()
 47940 timer_getoverrun()
 47941 timer_gettime()
 47942 timer_settime()
 47943 sleep()

47944 **RETURN VALUE**

47945 The *ualarm()* function shall return the number of microseconds remaining from the previous
 47946 *ualarm()* call. If no timeouts are pending or if *ualarm()* has not previously been called, *ualarm()*
 47947 shall return 0.

47948 **ERRORS**

47949 No errors are defined.

47950 **EXAMPLES**

47951 None.

47952 **APPLICATION USAGE**

47953 Applications are recommended to use *nanosleep()* if the Timers option is supported, or
 47954 *setitimer()*, *timer_create()*, *timer_delete()*, *timer_getoverrun()*, *timer_gettime()*, or *timer_settime()*
 47955 instead of this function.

47956 **RATIONALE**

47957 None.

47958 **FUTURE DIRECTIONS**

47959 None.

47960 **SEE ALSO**

47961 *alarm()*, *nanosleep()*, *setitimer()*, *sleep()*, *timer_create()*, *timer_delete()*, *timer_getoverrun()*, the Base
 47962 Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

47963 **CHANGE HISTORY**

47964 First released in Issue 4, Version 2.

47965 **Issue 5**

47966 Moved from X/OPEN UNIX extension to BASE.

47967 **Issue 6**

47968 This function is marked obsolescent.

47969 **NAME**47970 `ulimit` — get and set process limits47971 **SYNOPSIS**47972 xSI `#include <ulimit.h>`47973 `long ulimit(int cmd, ...);`

47974

47975 **DESCRIPTION**

47976 The `ulimit()` function shall control process limits. The process limits that can be controlled by
 47977 this function include the maximum size of a single file that can be written (this is equivalent to
 47978 using `setrlimit()` with `RLIMIT_FSIZE`). The `cmd` values, defined in `<ulimit.h>`, include:

47979 **UL_GETFSIZE** Return the file size limit (`RLIMIT_FSIZE`) of the process. The limit shall be in
 47980 units of 512-byte blocks and shall be inherited by child processes. Files of any
 47981 size can be read. The return value shall be the integer part of the soft file size
 47982 limit divided by 512. If the result cannot be represented as a **long**, the result is
 47983 unspecified.

47984 **UL_SETFSIZE** Set the file size limit for output operations of the process to the value of the
 47985 second argument, taken as a **long**, multiplied by 512. If the result would
 47986 overflow an **rlim_t**, the actual value set is unspecified. Any process may
 47987 decrease its own limit, but only a process with appropriate privileges may
 47988 increase the limit. The return value shall be the integer part of the new file size
 47989 limit divided by 512.

47990 The `ulimit()` function shall not change the setting of `errno` if successful.

47991 As all return values are permissible in a successful situation, an application wishing to check for
 47992 error situations should set `errno` to 0, then call `ulimit()`, and, if it returns `-1`, check to see if `errno` is
 47993 non-zero.

47994 **RETURN VALUE**

47995 Upon successful completion, `ulimit()` shall return the value of the requested limit. Otherwise, `-1`
 47996 shall be returned and `errno` set to indicate the error.

47997 **ERRORS**

47998 The `ulimit()` function shall fail and the limit shall be unchanged if:

47999 **[EINVAL]** The `cmd` argument is not valid.

48000 **[EPERM]** A process not having appropriate privileges attempts to increase its file size
 48001 limit.

48002 **EXAMPLES**

48003 None.

48004 **APPLICATION USAGE**

48005 None.

48006 **RATIONALE**

48007 None.

48008 **FUTURE DIRECTIONS**

48009 None.

48010 **SEE ALSO**

48011 *getrlimit()*, *setrlimit()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<ulimit.h>**

48012 **CHANGE HISTORY**

48013 First released in Issue 1. Derived from Issue 1 of the SVID.

48014 **Issue 5**

48015 In the description of `UL_SETFSIZE`, the text is corrected to refer to **`rlim_t`** rather than the
48016 spurious **`rlimit_t`**.

48017 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

48018 **NAME**

48019 umask — set and get the file mode creation mask

48020 **SYNOPSIS**

48021 #include <sys/stat.h>

48022 mode_t umask(mode_t *cmask*);48023 **DESCRIPTION**

48024 The *umask()* function shall set the process' file mode creation mask to *cmask* and return the
48025 previous value of the mask. Only the file permission bits of *cmask* (see <sys/stat.h>) are used; the
48026 meaning of the other bits is implementation-defined.

48027 The process' file mode creation mask is used during *open()*, *creat()*, *mkdir()*, and *mkfifo()* to turn
48028 off permission bits in the *mode* argument supplied. Bit positions that are set in *cmask* are cleared
48029 in the mode of the created file.

48030 **RETURN VALUE**

48031 The file permission bits in the value returned by *umask()* shall be the previous value of the file
48032 mode creation mask. The state of any other bits in that value is unspecified, except that a
48033 subsequent call to *umask()* with the returned value as *cmask* shall leave the state of the mask the
48034 same as its state before the first call, including any unspecified use of those bits.

48035 **ERRORS**

48036 No errors are defined.

48037 **EXAMPLES**

48038 None.

48039 **APPLICATION USAGE**

48040 None.

48041 **RATIONALE**

48042 Unsigned argument and return types for *umask()* were proposed. The return type and the
48043 argument were both changed to **mode_t**.

48044 Historical implementations have made use of additional bits in *cmask* for their implementation-
48045 defined purposes. The addition of the text that the meaning of other bits of the field is
48046 implementation-defined permits these implementations to conform to this volume of
48047 IEEE Std 1003.1-2001.

48048 **FUTURE DIRECTIONS**

48049 None.

48050 **SEE ALSO**

48051 *creat()*, *mkdir()*, *mkfifo()*, *open()*, the Base Definitions volume of IEEE Std 1003.1-2001,
48052 <sys/stat.h>, <sys/types.h>

48053 **CHANGE HISTORY**

48054 First released in Issue 1. Derived from Issue 1 of the SVID.

48055 **Issue 6**

48056 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

- 48057 The following new requirements on POSIX implementations derive from alignment with the
48058 Single UNIX Specification:
- 48059 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
48060 required for conforming implementations of previous POSIX specifications, it was not
48061 required for UNIX applications.

48062 **NAME**

48063 uname — get the name of the current system

48064 **SYNOPSIS**

48065 #include <sys/utsname.h>

48066 int uname(struct utsname *name);

48067 **DESCRIPTION**48068 The *uname()* function shall store information identifying the current system in the structure pointed to by *name*.48070 The *uname()* function uses the **utsname** structure defined in <sys/utsname.h>.

48071 The *uname()* function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

48076 The format of each member is implementation-defined.

48077 **RETURN VALUE**

48078 Upon successful completion, a non-negative value shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

48080 **ERRORS**

48081 No errors are defined.

48082 **EXAMPLES**

48083 None.

48084 **APPLICATION USAGE**

48085 The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.

48087 **RATIONALE**

48088 The values of the structure members are not constrained to have any relation to the version of this volume of IEEE Std 1003.1-2001 implemented in the operating system. An application should instead depend on *_POSIX_VERSION* and related constants defined in <unistd.h>.

48091 This volume of IEEE Std 1003.1-2001 does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.

48095 The *uname()* function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.

48098 4.3 BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric value, respectively. There are related *sethostname()* and *sethostid()* functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.

48102 **FUTURE DIRECTIONS**

48103 None.

48104 SEE ALSO

48105 The Base Definitions volume of IEEE Std 1003.1-2001, <**sys/utsname.h**>

48106 CHANGE HISTORY

48107 First released in Issue 1. Derived from Issue 1 of the SVID.

48108 **NAME**

48109 ungetc — push byte back into input stream

48110 **SYNOPSIS**

48111 #include <stdio.h>

48112 int ungetc(int *c*, FILE **stream*);48113 **DESCRIPTION**

48114 CX The functionality described on this reference page is aligned with the ISO C standard. Any
48115 conflict between the requirements described here and the ISO C standard is unintentional. This
48116 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

48117 The *ungetc()* function shall push the byte specified by *c* (converted to an **unsigned char**) back
48118 onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by
48119 subsequent reads on that stream in the reverse order of their pushing. A successful intervening
48120 call (with the stream pointed to by *stream*) to a file-positioning function (*fseek()*, *fsetpos()*, or
48121 *rewind()*) shall discard any pushed-back bytes for the stream. The external storage
48122 corresponding to the stream shall be unchanged.

48123 One byte of push-back shall be provided. If *ungetc()* is called too many times on the same stream
48124 without an intervening read or file-positioning operation on that stream, the operation may fail.

48125 If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall
48126 be left unchanged.

48127 A successful call to *ungetc()* shall clear the end-of-file indicator for the stream. The value of the
48128 file-position indicator for the stream after reading or discarding all pushed-back bytes shall be
48129 the same as it was before the bytes were pushed back. The file-position indicator is decremented
48130 by each successful call to *ungetc()*; if its value was 0 before a call, its value is unspecified after
48131 the call.

48132 **RETURN VALUE**

48133 Upon successful completion, *ungetc()* shall return the byte pushed back after conversion.
48134 Otherwise, it shall return EOF.

48135 **ERRORS**

48136 No errors are defined.

48137 **EXAMPLES**

48138 None.

48139 **APPLICATION USAGE**

48140 None.

48141 **RATIONALE**

48142 None.

48143 **FUTURE DIRECTIONS**

48144 None.

48145 **SEE ALSO**

48146 *fseek()*, *getc()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*, the Base Definitions volume of
48147 IEEE Std 1003.1-2001, <stdio.h>

48148 **CHANGE HISTORY**

48149 First released in Issue 1. Derived from Issue 1 of the SVID.

48150 **NAME**

48151 ungetwc — push wide-character code back into the input stream

48152 **SYNOPSIS**

48153 #include <stdio.h>

48154 #include <wchar.h>

48155 wint_t ungetwc(wint_t wc, FILE *stream);

48156 **DESCRIPTION**

48157 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 48158 conflict between the requirements described here and the ISO C standard is unintentional. This
 48159 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

48160 The *ungetwc()* function shall push the character corresponding to the wide-character code
 48161 specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters
 48162 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A
 48163 successful intervening call (with the stream pointed to by *stream*) to a file-positioning function
 48164 (*fseek()*, *fsetpos()*, or *rewind()*) discards any pushed-back characters for the stream. The external
 48165 storage corresponding to the stream is unchanged.

48166 At least one character of push-back shall be provided. If *ungetwc()* is called too many times on
 48167 the same stream without an intervening read or file-positioning operation on that stream, the
 48168 operation may fail.

48169 If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream
 48170 shall be left unchanged.

48171 A successful call to *ungetwc()* shall clear the end-of-file indicator for the stream. The value of the
 48172 file-position indicator for the stream after reading or discarding all pushed-back characters shall
 48173 be the same as it was before the characters were pushed back. The file-position indicator is
 48174 decremented (by one or more) by each successful call to *ungetwc()*; if its value was 0 before a
 48175 call, its value is unspecified after the call.

48176 **RETURN VALUE**

48177 Upon successful completion, *ungetwc()* shall return the wide-character code corresponding to
 48178 the pushed-back character. Otherwise, it shall return WEOF.

48179 **ERRORS**48180 The *ungetwc()* function may fail if:

48181 CX [EILSEQ] An invalid character sequence is detected, or a wide-character code does not
 48182 correspond to a valid character.

48183 **EXAMPLES**

48184 None.

48185 **APPLICATION USAGE**

48186 None.

48187 **RATIONALE**

48188 None.

48189 **FUTURE DIRECTIONS**

48190 None.

48191 **SEE ALSO**

48192 *fseek()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*, the Base Definitions volume of IEEE Std 1003.1-2001,
48193 **<stdio.h>**, **<wchar.h>**

48194 **CHANGE HISTORY**

48195 First released in Issue 4. Derived from the MSE working draft.

48196 **Issue 5**

48197 The Optional Header (OH) marking is removed from **<stdio.h>**.

48198 **Issue 6**

48199 The [EILSEQ] optional error condition is marked CX.

48200 NAME

48201 unlink — remove a directory entry

48202 SYNOPSIS

48203 #include <unistd.h>

48204 int unlink(const char *path);

48205 DESCRIPTION

48206 The *unlink()* function shall remove a link to a file. If *path* names a symbolic link, *unlink()* shall
 48207 remove the symbolic link named by *path* and shall not affect any file or directory named by the
 48208 contents of the symbolic link. Otherwise, *unlink()* shall remove the link named by the pathname
 48209 pointed to by *path* and shall decrement the link count of the file referenced by the link.

48210 When the file's link count becomes 0 and no process has the file open, the space occupied by the
 48211 file shall be freed and the file shall no longer be accessible. If one or more processes have the file
 48212 open when the last link is removed, the link shall be removed before *unlink()* returns, but the
 48213 removal of the file contents shall be postponed until all references to the file are closed.

48214 The *path* argument shall not name a directory unless the process has appropriate privileges and
 48215 the implementation supports using *unlink()* on directories.

48216 Upon successful completion, *unlink()* shall mark for update the *st_ctime* and *st_mtime* fields of
 48217 the parent directory. Also, if the file's link count is not 0, the *st_ctime* field of the file shall be
 48218 marked for update.

48219 RETURN VALUE

48220 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 48221 indicate the error. If -1 is returned, the named file shall not be changed.

48222 ERRORS

48223 The *unlink()* function shall fail and shall not unlink the file if:

48224 [EACCES] Search permission is denied for a component of the path prefix, or write
 48225 permission is denied on the directory containing the directory entry to be
 48226 removed.

48227 [EBUSY] The file named by the *path* argument cannot be unlinked because it is being
 48228 used by the system or another process and the implementation considers this
 48229 an error.

48230 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 48231 argument.

48232 [ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname
 48233 component is longer than {NAME_MAX}.
 48234

48235 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48236 [ENOTDIR] A component of the path prefix is not a directory.

48237 [EPERM] The file named by *path* is a directory, and either the calling process does not
 48238 have appropriate privileges, or the implementation prohibits using *unlink()*
 48239 on directories.

48240 XSI [EPERM] or [EACCES]
 48241 The S_ISVTX flag is set on the directory containing the file referred to by the
 48242 *path* argument and the caller is not the file owner, nor is the caller the
 48243 directory owner, nor does the caller have appropriate privileges.

48244 [EROFS] The directory entry to be unlinked is part of a read-only file system.

48245 The `unlink()` function may fail and not unlink the file if:

48246 XSI [EBUSY] The file named by *path* is a named STREAM.

48247 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
48248 resolution of the *path* argument.

48249 [ENAMETOOLONG]
48250 As a result of encountering a symbolic link in resolution of the *path* argument,
48251 the length of the substituted pathname string exceeded {PATH_MAX}.

48252 [ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared
48253 text) file that is being executed.

48254 EXAMPLES

48255 Removing a Link to a File

48256 The following example shows how to remove a link to a file named `/home/cnd/mod1` by
48257 removing the entry named `/modules/pass1`.

```
48258 #include <unistd.h>
48259 char *path = "/modules/pass1";
48260 int status;
48261 ...
48262 status = unlink(path);
```

48263 Checking for an Error

48264 The following example fragment creates a temporary password lock file named **LOCKFILE**,
48265 which is defined as `/etc/ptmp`, and gets a file descriptor for it. If the file cannot be opened for
48266 writing, `unlink()` is used to remove the link between the file descriptor and **LOCKFILE**.

```
48267 #include <sys/types.h>
48268 #include <stdio.h>
48269 #include <fcntl.h>
48270 #include <errno.h>
48271 #include <unistd.h>
48272 #include <sys/stat.h>
48273 #define LOCKFILE "/etc/ptmp"
48274 int pfd; /* Integer for file descriptor returned by open call. */
48275 FILE *fpfd; /* File pointer for use in putpwent(). */
48276 ...
48277 /* Open password Lock file. If it exists, this is an error. */
48278 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR
48279 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
48280     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
48281     exit(1);
48282 }
48283 /* Lock file created; proceed with fdopen of lock file so that
48284 putpwent() can be used.
48285 */
48286 if ((fpfd = fdopen(pfd, "w")) == NULL) {
```

```

48287         close(pfd);
48288         unlink(LOCKFILE);
48289         exit(1);
48290     }

```

48291 Replacing Files

48292 The following example fragment uses *unlink()* to discard links to files, so that they can be
 48293 replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error
 48294 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can
 48295 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```

48296     #include <sys/types.h>
48297     #include <stdio.h>
48298     #include <fcntl.h>
48299     #include <errno.h>
48300     #include <unistd.h>
48301     #include <sys/stat.h>

48302     #define LOCKFILE "/etc/ptmp"
48303     #define PASSWDFILE "/etc/passwd"
48304     #define SAVEFILE "/etc/opasswd"
48305     ...
48306     /* If no change was made, assume error and leave passwd unchanged. */
48307     if (!valid_change) {
48308         fprintf(stderr, "Could not change password for user %s\n", user);
48309         unlink(LOCKFILE);
48310         exit(1);
48311     }

48312     /* Change permissions on new password file. */
48313     chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

48314     /* Remove saved password file. */
48315     unlink(SAVEFILE);

48316     /* Save current password file. */
48317     link(PASSWDFILE, SAVEFILE);

48318     /* Remove current password file. */
48319     unlink(PASSWDFILE);

48320     /* Save new password file as current password file. */
48321     link(LOCKFILE, PASSWDFILE);

48322     /* Remove lock file. */
48323     unlink(LOCKFILE);

48324     exit(0);

```

48325 APPLICATION USAGE

48326 Applications should use *rmdir()* to remove a directory.

48327 RATIONALE

48328 Unlinking a directory is restricted to the superuser in many historical implementations for
 48329 reasons given in *link()* (see also *rename()*).

48330 The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume
48331 of IEEE Std 1003.1-2001 does not cover the system administration concepts of mounting and
48332 unmounting, the description of the error was changed to “resource busy”. (This meaning is used
48333 by some device drivers when a second process tries to open an exclusive use device.) The
48334 wording is also intended to allow implementations to refuse to remove a directory if it is the
48335 root or current working directory of any process.

48336 **FUTURE DIRECTIONS**

48337 None.

48338 **SEE ALSO**

48339 *close()*, *link()*, *remove()*, *rmdir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

48340 **CHANGE HISTORY**

48341 First released in Issue 1. Derived from Issue 1 of the SVID.

48342 **Issue 5**

48343 The [EBUSY] error is added to the “may fail” part of the ERRORS section.

48344 **Issue 6**

48345 The following new requirements on POSIX implementations derive from alignment with the
48346 Single UNIX Specification:

- 48347 • In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- 48348 • The [ELOOP] mandatory error condition is added.
- 48349 • A second [ENAMETOOLONG] is added as an optional error condition.
- 48350 • The [ETXTBSY] optional error condition is added.

48351 The following changes were made to align with the IEEE P1003.1a draft standard:

- 48352 • The [ELOOP] optional error condition is added.

48353 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48354 **NAME**

48355 unlockpt — unlock a pseudo-terminal master/slave pair

48356 **SYNOPSIS**48357 xSI `#include <stdlib.h>`48358 `int unlockpt(int fildev);`

48359

48360 **DESCRIPTION**48361 The *unlockpt()* function shall unlock the slave pseudo-terminal device associated with the
48362 master to which *fildev* refers.48363 Conforming applications shall ensure that they call *unlockpt()* before opening the slave side of a
48364 pseudo-terminal device.48365 **RETURN VALUE**48366 Upon successful completion, *unlockpt()* shall return 0. Otherwise, it shall return -1 and set *errno*
48367 to indicate the error.48368 **ERRORS**48369 The *unlockpt()* function may fail if:48370 [EBADF] The *fildev* argument is not a file descriptor open for writing.48371 [EINVAL] The *fildev* argument is not associated with a master pseudo-terminal device.48372 **EXAMPLES**

48373 None.

48374 **APPLICATION USAGE**

48375 None.

48376 **RATIONALE**

48377 None.

48378 **FUTURE DIRECTIONS**

48379 None.

48380 **SEE ALSO**48381 *grantpt()*, *open()*, *ptsname()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`48382 **CHANGE HISTORY**

48383 First released in Issue 4, Version 2.

48384 **Issue 5**

48385 Moved from X/OPEN UNIX extension to BASE.

48386 **Issue 6**

48387 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48388 **NAME**

48389 unsetenv — remove an environment variable

48390 **SYNOPSIS**48391 **CX** #include <stdlib.h>

48392 int unsetenv(const char *name);

48393

48394 **DESCRIPTION**

48395 The *unsetenv()* function shall remove an environment variable from the environment of the
 48396 calling process. The *name* argument points to a string, which is the name of the variable to be
 48397 removed. The named argument shall not contain an '=' character. If the named variable does
 48398 not exist in the current environment, the environment shall be unchanged and the function is
 48399 considered to have completed successfully.

48400 If the application modifies *environ* or the pointers to which it points, the behavior of *unsetenv()* is
 48401 undefined. The *unsetenv()* function shall update the list of pointers to which *environ* points.

48402 The *unsetenv()* function need not be reentrant. A function that is not required to be reentrant is
 48403 not required to be thread-safe.

48404 **RETURN VALUE**

48405 Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to
 48406 indicate the error, and the environment shall be unchanged.

48407 **ERRORS**48408 The *unsetenv()* function shall fail if:

48409 [EINVAL] The *name* argument is a null pointer, points to an empty string, or points to a
 48410 string containing an '=' character.

48411 **EXAMPLES**

48412 None.

48413 **APPLICATION USAGE**

48414 None.

48415 **RATIONALE**48416 Refer to the RATIONALE section in *setenv()*.48417 **FUTURE DIRECTIONS**

48418 None.

48419 **SEE ALSO**

48420 *getenv()*, *setenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>,
 48421 <sys/types.h>, <unistd.h>

48422 **CHANGE HISTORY**

48423 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

48424 **NAME**

48425 usleep — suspend execution for an interval

48426 **SYNOPSIS**

48427 OB XSI #include <unistd.h>

48428 int usleep(useconds_t useconds);

48429

48430 **DESCRIPTION**

48431 The *usleep()* function shall cause the calling thread to be suspended from execution until either
 48432 the number of realtime microseconds specified by the argument *useconds* has elapsed or a signal
 48433 is delivered to the calling thread and its action is to invoke a signal-catching function or to
 48434 terminate the process. The suspension time may be longer than requested due to the scheduling
 48435 of other activity by the system.

48436 The *useconds* argument shall be less than one million. If the value of *useconds* is 0, then the call
 48437 has no effect.

48438 If a SIGALRM signal is generated for the calling process during execution of *usleep()* and if the
 48439 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *usleep()*
 48440 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also
 48441 unspecified whether it remains pending after *usleep()* returns or it is discarded.

48442 If a SIGALRM signal is generated for the calling process during execution of *usleep()*, except as a
 48443 result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from
 48444 delivery, it is unspecified whether that signal has any effect other than causing *usleep()* to return.

48445 If a signal-catching function interrupts *usleep()* and examines or changes either the time a
 48446 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or
 48447 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

48448 If a signal-catching function interrupts *usleep()* and calls *siglongjmp()* or *longjmp()* to restore an
 48449 environment saved prior to the *usleep()* call, the action associated with the SIGALRM signal and
 48450 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also
 48451 unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored
 48452 as part of the environment.

48453 Implementations may place limitations on the granularity of timer values. For each interval
 48454 timer, if the requested timer value requires a finer granularity than the implementation supports,
 48455 the actual timer value shall be rounded up to the next supported value.

48456 Interactions between *usleep()* and any of the following are unspecified:

48457 *nanosleep()*
 48458 *setitimer()*
 48459 *timer_create()*
 48460 *timer_delete()*
 48461 *timer_getoverrun()*
 48462 *timer_gettime()*
 48463 *timer_settime()*
 48464 *ualarm()*
 48465 *sleep()*

48466 RETURN VALUE

48467 Upon successful completion, *usleep()* shall return 0; otherwise, it shall return -1 and set *errno* to
48468 indicate the error.

48469 ERRORS

48470 The *usleep()* function may fail if:

48471 [EINVAL] The time interval specified one million or more microseconds.

48472 EXAMPLES

48473 None.

48474 APPLICATION USAGE

48475 Applications are recommended to use *nanosleep()* if the Timers option is supported, or
48476 *setitimer()*, *timer_create()*, *timer_delete()*, *timer_getoverrun()*, *timer_gettime()*, or *timer_settime()*
48477 instead of this function.

48478 RATIONALE

48479 None.

48480 FUTURE DIRECTIONS

48481 None.

48482 SEE ALSO

48483 *alarm()*, *getitimer()*, *nanosleep()*, *sigaction()*, *sleep()*, *timer_create()*, *timer_delete()*,
48484 *timer_getoverrun()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

48485 CHANGE HISTORY

48486 First released in Issue 4, Version 2.

48487 Issue 5

48488 Moved from X/OPEN UNIX extension to BASE.

48489 The DESCRIPTION is changed to indicate that timers are now thread-based rather than
48490 process-based.

48491 Issue 6

48492 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48493 This function is marked obsolescent.

NAME

utime — set file access and modification times

SYNOPSIS

```
#include <utime.h>
```

```
int utime(const char *path, const struct utimbuf *times);
```

DESCRIPTION

The *utime()* function shall set the access and modification times of the file named by the *path* argument.

If *times* is a null pointer, the access and modification times of the file shall be set to the current time. The effective user ID of the process shall match the owner of the file, or the process has write permission to the file or has appropriate privileges, to use *utime()* in this manner.

If *times* is not a null pointer, *times* shall be interpreted as a pointer to a **utimbuf** structure and the access and modification times shall be set to the values contained in the designated structure. Only a process with the effective user ID equal to the user ID of the file or a process with appropriate privileges may use *utime()* this way.

The **utimbuf** structure is defined in the **<utime.h>** header. The times in the structure **utimbuf** are measured in seconds since the Epoch.

Upon successful completion, *utime()* shall mark the time of the last file status change, *st_ctime*, to be updated; see **<sys/stat.h>**.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall be set to indicate the error, and the file times shall not be affected.

ERRORS

The *utime()* function shall fail if:

[EACCES] Search permission is denied by a component of the path prefix; or the *times* argument is a null pointer and the effective user ID of the process does not match the owner of the file, the process does not have write permission for the file, and the process does not have appropriate privileges.

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

[ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

[ENOTDIR] A component of the path prefix is not a directory.

[EPERM] The *times* argument is not a null pointer and the calling process' effective user ID does not match the owner of the file and the calling process does not have the appropriate privileges.

[EROFS] The file system containing the file is read-only.

The *utime()* function may fail if:

[ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

48536 [ENAMETOOLONG]

48537 As a result of encountering a symbolic link in resolution of the *path* argument,
48538 the length of the substituted pathname string exceeded {PATH_MAX}.

48539 **EXAMPLES**

48540 None.

48541 **APPLICATION USAGE**

48542 None.

48543 **RATIONALE**

48544 The *actime* structure member must be present so that an application may set it, even though an
48545 implementation may ignore it and not change the access time on the file. If an application
48546 intends to leave one of the times of a file unchanged while changing the other, it should use
48547 *stat()* to retrieve the file's *st_atime* and *st_mtime* parameters, set *actime* and *modtime* in the buffer,
48548 and change one of them before making the *utime()* call.

48549 **FUTURE DIRECTIONS**

48550 None.

48551 **SEE ALSO**

48552 The Base Definitions volume of IEEE Std 1003.1-2001, <**sys/stat.h**>, <**utime.h**>

48553 **CHANGE HISTORY**

48554 First released in Issue 1. Derived from Issue 1 of the SVID.

48555 **Issue 6**

48556 The following new requirements on POSIX implementations derive from alignment with the
48557 Single UNIX Specification:

- 48558 • The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was
48559 required for conforming implementations of previous POSIX specifications, it was not
48560 required for UNIX applications.
- 48561 • The [ELOOP] mandatory error condition is added.
- 48562 • A second [ENAMETOOLONG] is added as an optional error condition.

48563 The following changes were made to align with the IEEE P1003.1a draft standard:

- 48564 • The [ELOOP] optional error condition is added.

48565 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48566 **NAME**48567 utimes — set file access and modification times (**LEGACY**)48568 **SYNOPSIS**48569 xSI `#include <sys/time.h>`48570 `int utimes(const char *path, const struct timeval times[2]);`

48571

48572 **DESCRIPTION**

48573 The *utimes()* function shall set the access and modification times of the file pointed to by the *path*
 48574 argument to the value of the *times* argument. The *utimes()* function allows time specifications
 48575 accurate to the microsecond.

48576 For *utimes()*, the *times* argument is an array of **timeval** structures. The first array member
 48577 represents the date and time of last access, and the second member represents the date and time
 48578 of last modification. The times in the **timeval** structure are measured in seconds and
 48579 microseconds since the Epoch, although rounding toward the nearest second may occur.

48580 If the *times* argument is a null pointer, the access and modification times of the file shall be set to
 48581 the current time. The effective user ID of the process shall match the owner of the file, or has
 48582 write access to the file or appropriate privileges to use this call in this manner. Upon completion,
 48583 *utimes()* shall mark the time of the last file status change, *st_ctime*, for update.

48584 **RETURN VALUE**

48585 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall
 48586 be set to indicate the error, and the file times shall not be affected.

48587 **ERRORS**48588 The *utimes()* function shall fail if:

48589 [EACCES] Search permission is denied by a component of the path prefix; or the *times*
 48590 argument is a null pointer and the effective user ID of the process does not
 48591 match the owner of the file and write access is denied.

48592 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 48593 argument.

48594 [ENAMETOOLONG]
 48595 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 48596 component is longer than {NAME_MAX}.

48597 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48598 [ENOTDIR] A component of the path prefix is not a directory.

48599 [EPERM] The *times* argument is not a null pointer and the calling process' effective user
 48600 ID has write access to the file but does not match the owner of the file and the
 48601 calling process does not have the appropriate privileges.

48602 [EROFS] The file system containing the file is read-only.

48603 The *utimes()* function may fail if:

48604 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 48605 resolution of the *path* argument.

48606 [ENAMETOOLONG]
 48607 Pathname resolution of a symbolic link produced an intermediate result
 48608 whose length exceeds {PATH_MAX}.

48609 **EXAMPLES**

48610 None.

48611 **APPLICATION USAGE**

48612 For applications portability, the *utime()* function should be used to set file access and
48613 modification times instead of *utimes()*.

48614 **RATIONALE**

48615 None.

48616 **FUTURE DIRECTIONS**

48617 This function may be withdrawn in a future version.

48618 **SEE ALSO**48619 *utime()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/time.h>48620 **CHANGE HISTORY**

48621 First released in Issue 4, Version 2.

48622 **Issue 5**

48623 Moved from X/OPEN UNIX extension to BASE.

48624 **Issue 6**

48625 This function is marked LEGACY.

48626 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48627 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
48628 [ELOOP] error condition is added.

48629 **NAME**

48630 va_arg, va_copy, va_end, va_start — handle variable argument list

48631 **SYNOPSIS**

48632 #include <stdarg.h>

48633 type va_arg(va_list ap, type);

48634 void va_copy(va_list dest, va_list src);

48635 void va_end(va_list ap);

48636 void va_start(va_list ap, argN);

48637 **DESCRIPTION**

48638 Refer to the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>.

48639 **NAME**

48640 vfork — create a new process; share virtual memory

48641 **SYNOPSIS**48642 OB XSI `#include <unistd.h>`48643 `pid_t vfork(void);`

48644

48645 **DESCRIPTION**

48646 The `vfork()` function shall be equivalent to `fork()`, except that the behavior is undefined if the
 48647 process created by `vfork()` either modifies any data other than a variable of type **pid_t** used to
 48648 store the return value from `vfork()`, or returns from the function in which `vfork()` was called, or
 48649 calls any other function before successfully calling `_exit()` or one of the *exec* family of functions.

48650 **RETURN VALUE**

48651 Upon successful completion, `vfork()` shall return 0 to the child process and return the process ID
 48652 of the child process to the parent process. Otherwise, `-1` shall be returned to the parent, no child
 48653 process shall be created, and *errno* shall be set to indicate the error.

48654 **ERRORS**48655 The `vfork()` function shall fail if:

48656 [EAGAIN] The system-wide limit on the total number of processes under execution
 48657 would be exceeded, or the system-imposed limit on the total number of
 48658 processes under execution by a single user would be exceeded.

48659 [ENOMEM] There is insufficient swap space for the new process.

48660 **EXAMPLES**

48661 None.

48662 **APPLICATION USAGE**

48663 Conforming applications are recommended not to depend on `vfork()`, but to use `fork()` instead.
 48664 The `vfork()` function may be withdrawn in a future version.

48665 On some implementations, `vfork()` is equivalent to `fork()`.

48666 The `vfork()` function differs from `fork()` only in that the child process can share code and data
 48667 with the calling process (parent process). This speeds cloning activity significantly at a risk to
 48668 the integrity of the parent process if `vfork()` is misused.

48669 The use of `vfork()` for any purpose except as a prelude to an immediate call to a function from
 48670 the *exec* family, or to `_exit()`, is not advised.

48671 The `vfork()` function can be used to create new processes without fully copying the address
 48672 space of the old process. If a forked process is simply going to call *exec*, the data space copied
 48673 from the parent to the child by `fork()` is not used. This is particularly inefficient in a paged
 48674 environment, making `vfork()` particularly useful. Depending upon the size of the parent's data
 48675 space, `vfork()` can give a significant performance improvement over `fork()`.

48676 The `vfork()` function can normally be used just like `fork()`. It does not work, however, to return
 48677 while running in the child's context from the caller of `vfork()` since the eventual return from
 48678 `vfork()` would then return to a no longer existent stack frame. Care should be taken, also, to call
 48679 `_exit()` rather than `exit()` if *exec* cannot be used, since `exit()` flushes and closes standard I/O
 48680 channels, thereby damaging the parent process' standard I/O data structures. (Even with `fork()`,
 48681 it is wrong to call `exit()`, since buffered data would then be flushed twice.)

48682 If signal handlers are invoked in the child process after `vfork()`, they must follow the same rules
 48683 as other code in the child process.

48684 **RATIONALE**

48685 None.

48686 **FUTURE DIRECTIONS**

48687 This function may be withdrawn in a future version.

48688 **SEE ALSO**48689 *exec*, *exit()*, *fork()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>48690 **CHANGE HISTORY**

48691 First released in Issue 4, Version 2.

48692 **Issue 5**

48693 Moved from X/OPEN UNIX extension to BASE.

48694 **Issue 6**

48695 This function is marked obsolescent.

48696 **NAME**

48697 vfprintf, vprintf, vsnprintf, vsprintf — format output of a stdarg argument list

48698 **SYNOPSIS**

48699 #include <stdarg.h>

48700 #include <stdio.h>

48701 int vfprintf(FILE *restrict *stream*, const char *restrict *format*,
48702 va_list *ap*);48703 int vprintf(const char *restrict *format*, va_list *ap*);48704 int vsnprintf(char *restrict *s*, size_t *n*, const char *restrict *format*,
48705 va_list *ap*);48706 int vsprintf(char *restrict *s*, const char *restrict *format*, va_list *ap*);48707 **DESCRIPTION**48708 CX The functionality described on this reference page is aligned with the ISO C standard. Any
48709 conflict between the requirements described here and the ISO C standard is unintentional. This
48710 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.48711 The *vprintf()*, *vfprintf()*, *vsnprintf()*, and *vsprintf()* functions shall be equivalent to *printf()*,
48712 *fprintf()*, *snprintf()*, and *sprintf()* respectively, except that instead of being called with a variable
48713 number of arguments, they are called with an argument list as defined by <stdarg.h>.48714 These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro,
48715 the value of *ap* after the return is unspecified.48716 **RETURN VALUE**48717 Refer to *fprintf()*.48718 **ERRORS**48719 Refer to *fprintf()*.48720 **EXAMPLES**

48721 None.

48722 **APPLICATION USAGE**48723 Applications using these functions should call *va_end(ap)* afterwards to clean up.48724 **RATIONALE**

48725 None.

48726 **FUTURE DIRECTIONS**

48727 None.

48728 **SEE ALSO**48729 *fprintf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>, <stdio.h>48730 **CHANGE HISTORY**

48731 First released in Issue 1. Derived from Issue 1 of the SVID.

48732 **Issue 5**48733 The *vsnprintf()* function is added.48734 **Issue 6**48735 The *vfprintf()*, *vprintf()*, *vsnprintf()*, and *vsprintf()* functions are updated for alignment with the
48736 ISO / IEC 9899:1999 standard.

48737 NAME

48738 vfscanf, vscanf, vsscanf — format input of a stdarg argument list

48739 SYNOPSIS

48740 #include <stdarg.h>

48741 #include <stdio.h>

48742 int vfscanf(FILE *restrict stream, const char *restrict format,

48743 va_list arg);

48744 int vscanf(const char *restrict format, va_list arg);

48745 int vsscanf(const char *restrict s, const char *restrict format,

48746 va_list arg);

48747 DESCRIPTION

48748 CX The functionality described on this reference page is aligned with the ISO C standard. Any
48749 conflict between the requirements described here and the ISO C standard is unintentional. This
48750 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

48751 The *vscanf()*, *vfscanf()*, and *vsscanf()* functions shall be equivalent to the *scanf()*, *fscanf()*, and
48752 *sscanf()* functions, respectively, except that instead of being called with a variable number of
48753 arguments, they are called with an argument list as defined in the <stdarg.h> header. These
48754 functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the
48755 value of *ap* after the return is unspecified.

48756 RETURN VALUE

48757 Refer to *fscanf()*.

48758 ERRORS

48759 Refer to *fscanf()*.

48760 EXAMPLES

48761 None.

48762 APPLICATION USAGE

48763 Applications using these functions should call *va_end(ap)* afterwards to clean up.

48764 RATIONALE

48765 None.

48766 FUTURE DIRECTIONS

48767 None.

48768 SEE ALSO

48769 *fscanf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>, <stdio.h>

48770 CHANGE HISTORY

48771 First released in Issue 6. Derived from the ISO / IEC 9899:1999 standard.

48772 **NAME**

48773 vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

48774 **SYNOPSIS**

48775 #include <stdarg.h>

48776 #include <stdio.h>

48777 #include <wchar.h>

48778 int vfwprintf(FILE *restrict *stream*, const wchar_t *restrict *format*,
48779 va_list *arg*);48780 int vswprintf(wchar_t *restrict *ws*, size_t *n*,48781 const wchar_t *restrict *format*, va_list *arg*);48782 int vwprintf(const wchar_t *restrict *format*, va_list *arg*);48783 **DESCRIPTION**48784 CX The functionality described on this reference page is aligned with the ISO C standard. Any
48785 conflict between the requirements described here and the ISO C standard is unintentional. This
48786 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.48787 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* functions shall be equivalent to *fwprintf()*, *swprintf()*,
48788 and *wprintf()* respectively, except that instead of being called with a variable number of
48789 arguments, they are called with an argument list as defined by <stdarg.h>.48790 These functions shall not invoke the *va_end* macro. However, as these functions do invoke the
48791 *va_arg* macro, the value of *ap* after the return is unspecified.48792 **RETURN VALUE**48793 Refer to *fwprintf()*.48794 **ERRORS**48795 Refer to *fwprintf()*.48796 **EXAMPLES**

48797 None.

48798 **APPLICATION USAGE**48799 Applications using these functions should call *va_end(ap)* afterwards to clean up.48800 **RATIONALE**

48801 None.

48802 **FUTURE DIRECTIONS**

48803 None.

48804 **SEE ALSO**48805 *fwprintf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>, <stdio.h>,
48806 <wchar.h>48807 **CHANGE HISTORY**48808 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
48809 (E).48810 **Issue 6**48811 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* prototypes are updated for alignment with the
48812 ISO/IEC 9899:1999 standard. ()

48813 **NAME**

48814 vfwscanf, vswscanf, vwscanf — wide-character formatted input of a stdarg argument list

48815 **SYNOPSIS**

48816 #include <stdarg.h>

48817 #include <stdio.h>

48818 #include <wchar.h>

48819 int vfwscanf(FILE *restrict *stream*, const wchar_t *restrict *format*,
48820 va_list *arg*);48821 int vswscanf(const wchar_t *restrict *ws*, const wchar_t *restrict *format*,
48822 va_list *arg*);48823 int vwscanf(const wchar_t *restrict *format*, va_list *arg*);48824 **DESCRIPTION**48825 CX The functionality described on this reference page is aligned with the ISO C standard. Any
48826 conflict between the requirements described here and the ISO C standard is unintentional. This
48827 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.48828 The *vfwscanf()*, *vswscanf()*, and *vwscanf()* functions shall be equivalent to the *fwscanf()*,
48829 *swscanf()*, and *wscanf()* functions, respectively, except that instead of being called with a
48830 variable number of arguments, they are called with an argument list as defined in the <stdarg.h>
48831 header. These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg*
48832 macro, the value of *ap* after the return is unspecified.48833 **RETURN VALUE**48834 Refer to *fwscanf()*.48835 **ERRORS**48836 Refer to *fwscanf()*.48837 **EXAMPLES**

48838 None.

48839 **APPLICATION USAGE**48840 Applications using these functions should call *va_end(ap)* afterwards to clean up.48841 **RATIONALE**

48842 None.

48843 **FUTURE DIRECTIONS**

48844 None.

48845 **SEE ALSO**48846 *fwscanf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>, <stdio.h>,
48847 <wchar.h>48848 **CHANGE HISTORY**

48849 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

48850 **NAME**

48851 vprintf — format the output of a stdarg argument list

48852 **SYNOPSIS**

48853 #include <stdarg.h>

48854 #include <stdio.h>

48855 int vprintf(const char *restrict *format*, va_list *ap*);48856 **DESCRIPTION**48857 Refer to *vfprintf()*.

48858 **NAME**

48859 vscanf — format input of a stdarg argument list

48860 **SYNOPSIS**

48861 #include <stdarg.h>

48862 #include <stdio.h>

48863 int vscanf(const char *restrict *format*, va_list *arg*);48864 **DESCRIPTION**48865 Refer to *vscanf()*.

48866 **NAME**

48867 vsprintf, vsprintf — format output of a stdarg argument list

48868 **SYNOPSIS**

48869 #include <stdarg.h>

48870 #include <stdio.h>

48871 int vsnprintf(char *restrict *s*, size_t *n*,48872 const char *restrict *format*, va_list *ap*);48873 int vsprintf(char *restrict *s*, const char *restrict *format*,48874 va_list *ap*);48875 **DESCRIPTION**48876 Refer to *vfprintf()*.

48877 **NAME**

48878 vsscanf — format input of a stdarg argument list

48879 **SYNOPSIS**

48880 #include <stdarg.h>

48881 #include <stdio.h>

48882 int vsscanf(const char *restrict *s*, const char *restrict *format*,48883 va_list *arg*);48884 **DESCRIPTION**48885 Refer to *vfscanf()*.

48886 **NAME**

48887 vswprintf — wide-character formatted output of a stdarg argument list

48888 **SYNOPSIS**

48889 #include <stdarg.h>

48890 #include <stdio.h>

48891 #include <wchar.h>

```
48892       int vswprintf(wchar_t *restrict ws, size_t n,  
48893                    const wchar_t *restrict format, va_list arg);
```

48894 **DESCRIPTION**48895 Refer to *vfwprintf()*.

48896 **NAME**

48897 vswscanf — wide-character formatted input of a stdarg argument list

48898 **SYNOPSIS**

48899 #include <stdarg.h>

48900 #include <stdio.h>

48901 #include <wchar.h>

48902 int vswscanf(const wchar_t *restrict *ws*, const wchar_t *restrict *format*,48903 va_list *arg*);48904 **DESCRIPTION**48905 Refer to *vfwscanf()*.

48906 **NAME**

48907 vwprintf — wide-character formatted output of a stdarg argument list

48908 **SYNOPSIS**

48909 #include <stdarg.h>

48910 #include <stdio.h>

48911 #include <wchar.h>

48912 int vwprintf(const wchar_t *restrict *format*, va_list *arg*);48913 **DESCRIPTION**48914 Refer to *vfwprintf()*.

48915 **NAME**

48916 vwscanf — wide-character formatted input of a stdarg argument list

48917 **SYNOPSIS**

48918 #include <stdarg.h>

48919 #include <stdio.h>

48920 #include <wchar.h>

48921 int vwscanf(const wchar_t *restrict *format*, va_list *arg*);48922 **DESCRIPTION**48923 Refer to *vfwscanf()*.

48924 **NAME**

48925 wait, waitpid — wait for a child process to stop or terminate

48926 **SYNOPSIS**

48927 #include <sys/wait.h>

48928 pid_t wait(int *stat_loc);

48929 pid_t waitpid(pid_t pid, int *stat_loc, int options);

48930 **DESCRIPTION**

48931 The *wait()* and *waitpid()* functions shall obtain status information pertaining to one of the
 48932 caller's child processes. Various options permit status information to be obtained for child
 48933 processes that have terminated or stopped. If status information is available for two or more
 48934 child processes, the order in which their status is reported is unspecified.

48935 The *wait()* function shall suspend execution of the calling thread until status information for one
 48936 of the terminated child processes of the calling process is available, or until delivery of a signal
 48937 whose action is either to execute a signal-catching function or to terminate the process. If more
 48938 than one thread is suspended in *wait()* or *waitpid()* awaiting termination of the same process,
 48939 exactly one thread shall return the process status at the time of the target process termination. If
 48940 status information is available prior to the call to *wait()*, return shall be immediate.

48941 The *waitpid()* function shall be equivalent to *wait()* if the *pid* argument is (**pid_t**)−1 and the
 48942 *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and
 48943 *options* arguments.

48944 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()*
 48945 function shall only return the status of a child process from this set:

- 48946 • If *pid* is equal to (**pid_t**)−1, *status* is requested for any child process. In this respect, *waitpid()*
 48947 is then equivalent to *wait()*.
- 48948 • If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is
 48949 requested.
- 48950 • If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of
 48951 the calling process.
- 48952 • If *pid* is less than (**pid_t**)−1, *status* is requested for any child process whose process group ID
 48953 is equal to the absolute value of *pid*.

48954 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the
 48955 following flags, defined in the <sys/wait.h> header:

48956 XSI **WCONTINUED** The *waitpid()* function shall report the status of any continued child process
 48957 specified by *pid* whose status has not been reported since it continued from a
 48958 job control stop.

48959 **WNOHANG** The *waitpid()* function shall not suspend execution of the calling thread if
 48960 *status* is not immediately available for one of the child processes specified by
 48961 *pid*.

48962 **WUNTRACED** The status of any child processes specified by *pid* that are stopped, and whose
 48963 status has not yet been reported since they stopped, shall also be reported to
 48964 the requesting process.

48965 XSI If the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, and the
 48966 process has no unwaited-for children that were transformed into zombie processes, the calling
 48967 thread shall block until all of the children of the process containing the calling thread terminate,
 48968 and *wait()* and *waitpid()* shall fail and set *errno* to [ECHILD].

If *wait()* or *waitpid()* return because the status of a child process is available, these functions shall return a value equal to the process ID of the child process. In this case, if the value of the argument *stat_loc* is not a null pointer, information shall be stored in the location pointed to by *stat_loc*. The value stored at the location pointed to by *stat_loc* shall be 0 if and only if the status returned is from a terminated child process that terminated by one of the following means:

1. The process returned 0 from *main()*.
2. The process called *_exit()* or *exit()* with a *status* argument of 0.
3. The process was terminated because the last thread in the process terminated.

Regardless of its value, this information may be interpreted using the following macros, which are defined in **<sys/wait.h>** and evaluate to integral expressions; the *stat_val* argument is the integer value pointed to by *stat_loc*.

WIFEXITED(*stat_val*)

Evaluates to a non-zero value if *status* was returned for a child process that terminated normally.

WEXITSTATUS(*stat_val*)

If the value of **WIFEXITED(*stat_val*)** is non-zero, this macro evaluates to the low-order 8 bits of the *status* argument that the child process passed to *_exit()* or *exit()*, or the value the child process returned from *main()*.

WIFSIGNALED(*stat_val*)

Evaluates to a non-zero value if *status* was returned for a child process that terminated due to the receipt of a signal that was not caught (see **<signal.h>**).

WTERMSIG(*stat_val*)

If the value of **WIFSIGNALED(*stat_val*)** is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.

WIFSTOPPED(*stat_val*)

Evaluates to a non-zero value if *status* was returned for a child process that is currently stopped.

WSTOPSIG(*stat_val*)

If the value of **WIFSTOPPED(*stat_val*)** is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.

WIFCONTINUED(*stat_val*)

Evaluates to a non-zero value if *status* was returned for a child process that has continued from a job control stop.

It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes created by *posix_spawn()* or *posix_spawnnp()* can indicate a **WIFSTOPPED(*stat_val*)** before subsequent calls to *wait()* or *waitpid()* indicate **WIFEXITED(*stat_val*)** as the result of an error detected before the new process image starts executing.

It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes created by *posix_spawn()* or *posix_spawnnp()* can indicate a **WIFSIGNALED(*stat_val*)** if a signal is sent to the parent's process group after *posix_spawn()* or *posix_spawnnp()* is called.

If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that specified the **WUNTRACED** flag and did not specify the **WCONTINUED** flag, exactly one of the macros **WIFEXITED(**stat_loc*)**, **WIFSIGNALED(**stat_loc*)**, and **WIFSTOPPED(**stat_loc*)** shall evaluate to a non-zero value.

49013 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that specified the
 49014 XSI WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(**stat_loc*),
 49015 XSI WIFSIGNALED(**stat_loc*), WIFSTOPPED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall
 49016 evaluate to a non-zero value.

49017 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that did not specify the
 49018 XSI WUNTRACED or WCONTINUED flags, or by a call to the *wait()* function, exactly one of the
 49019 macros WIFEXITED(**stat_loc*) and WIFSIGNALED(**stat_loc*) shall evaluate to a non-zero value.

49020 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that did not specify the
 49021 XSI WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait()* function,
 49022 XSI exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and
 49023 WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

49024 If _POSIX_REALTIME_SIGNALS is defined, and the implementation queues the SIGCHLD
 49025 signal, then if *wait()* or *waitpid()* returns because the status of a child process is available, any
 49026 pending SIGCHLD signal associated with the process ID of the child process shall be discarded.
 49027 Any other pending SIGCHLD signals shall remain pending.

49028 Otherwise, if SIGCHLD is blocked, if *wait()* or *waitpid()* return because the status of a child
 49029 process is available, any pending SIGCHLD signal shall be cleared unless the status of another
 49030 child process is available.

49031 For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD
 49032 signal is delivered.

49033 There may be additional implementation-defined circumstances under which *wait()* or *waitpid()*
 49034 report *status*. This shall not occur unless the calling process or one of its child processes explicitly
 49035 makes use of a non-standard extension. In these cases the interpretation of the reported *status* is
 49036 implementation-defined.

49037 XSI If a parent process terminates without waiting for all of its child processes to terminate, the
 49038 remaining child processes shall be assigned a new parent process ID corresponding to an
 49039 implementation-defined system process.

49040 RETURN VALUE

49041 If *wait()* or *waitpid()* returns because the status of a child process is available, these functions
 49042 shall return a value equal to the process ID of the child process for which *status* is reported. If
 49043 *wait()* or *waitpid()* returns due to the delivery of a signal to the calling process, -1 shall be
 49044 returned and *errno* set to [EINTR]. If *waitpid()* was invoked with WNOHANG set in *options*, it
 49045 has at least one child process specified by *pid* for which *status* is not available, and *status* is not
 49046 available for any process specified by *pid*, 0 is returned. Otherwise, (*pid_t*)-1 shall be returned,
 49047 and *errno* set to indicate the error.

49048 ERRORS

49049 The *wait()* function shall fail if:

49050 [ECHILD] The calling process has no existing unwaited-for child processes.

49051 [EINTR] The function was interrupted by a signal. The value of the location pointed to
 49052 by *stat_loc* is undefined.

49053 The *waitpid()* function shall fail if:

49054 [ECHILD] The process specified by *pid* does not exist or is not a child of the calling
 49055 process, or the process group specified by *pid* does not exist or does not have
 49056 any member process that is a child of the calling process.

49057 [EINTR] The function was interrupted by a signal. The value of the location pointed to
 49058 by *stat_loc* is undefined.

49059 [EINVAL] The *options* argument is not valid.

49060 EXAMPLES

49061 None.

49062 APPLICATION USAGE

49063 None.

49064 RATIONALE

49065 A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the
 49066 calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an
 49067 *exec* or other function calls) from the parent. If a child produces grandchildren by further use of
 49068 *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()*
 49069 from the original parent process. Nothing in this volume of IEEE Std 1003.1-2001 prevents an
 49070 implementation from providing extensions that permit a process to get *status* from a grandchild
 49071 or any other process, but a process that does not use such extensions must be guaranteed to see
 49072 *status* from only its direct children.

49073 The *waitpid()* function is provided for three reasons:

- 49074 1. To support job control
- 49075 2. To permit a non-blocking version of the *wait()* function
- 49076 3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without
 49077 interfering with other terminated children for which the process has not waited

49078 The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The
 49079 function uses the *options* argument, which is equivalent to an argument to *wait3()*. The
 49080 WUNTRACED flag is used only in conjunction with job control on systems supporting job
 49081 control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped
 49082 processes in that implementation: processes being traced via the *ptrace()* debugging facility and
 49083 (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of
 49084 IEEE Std 1003.1-2001, only the second type is relevant. The name WUNTRACED was retained
 49085 because its usage is the same, even though the name is not intuitively meaningful in this context.

49086 The third reason for the *waitpid()* function is to permit independent sections of a process to
 49087 spawn and wait for children without interfering with each other. For example, the following
 49088 problem occurs in developing a portable shell, or command interpreter:

```
49089 stream = popen("/bin/true");
49090 (void) system("sleep 100");
49091 (void) pclose(stream);
```

49092 On all historical implementations, the final *pclose()* fails to reap the *wait()* *status* of the *popen()*.

49093 The status values are retrieved by macros, rather than given as specific bit encodings as they are
 49094 in most historical implementations (and thus expected by existing programs). This was
 49095 necessary to eliminate a limitation on the number of signals an implementation can support that
 49096 was inherent in the traditional encodings. This volume of IEEE Std 1003.1-2001 does require that
 49097 a *status* value of zero corresponds to a process calling *_exit(0)*, as this is the most common
 49098 encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

49099 These macros syntactically operate on an arbitrary integer value. The behavior is undefined
 49100 unless that value is one stored by a successful call to *wait()* or *waitpid()* in the location pointed
 49101 to by the *stat_loc* argument. An early proposal attempted to make this clearer by specifying each

argument as **stat_loc* rather than *stat_val*. However, that did not follow the conventions of other specifications in this volume of IEEE Std 1003.1-2001 or traditional usage. It also could have implied that the argument to the macro must literally be **stat_loc*; in fact, that value can be stored or passed as an argument to other functions before being interpreted by these macros.

The extension that affects *wait()* and *waitpid()* and is common in historical implementations is the *ptrace()* function. It is called by a child process and causes that child to stop and return a *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace()* children is traditionally returned regardless of the WUNTRACED flag (or by the *wait()* function). Most applications do not need to concern themselves with such extensions because they have control over what extensions they or their children use. However, applications, such as command interpreters, that invoke arbitrary processes may see this behavior when those arbitrary processes misuse such extensions.

Implementations that support **core** file creation or other implementation-defined actions on termination of some processes traditionally provide a bit in the *status* returned by *wait()* to indicate that such actions have occurred.

Allowing the *wait()* family of functions to discard a pending SIGCHLD signal that is associated with a successfully waited-for child process puts them into the *sigwait()* and *sigwaitinfo()* category with respect to SIGCHLD.

This definition allows implementations to treat a pending SIGCHLD signal as accepted by the process in *wait()*, with the same meaning of “accepted” as when that word is applied to the *sigwait()* family of functions.

Allowing the *wait()* family of functions to behave this way permits an implementation to be able to deal precisely with SIGCHLD signals.

In particular, an implementation that does accept (discard) the SIGCHLD signal can make the following guarantees regardless of the queuing depth of signals in general (the list of waitable children can hold the SIGCHLD queue):

1. If a SIGCHLD signal handler is established via *sigaction()* without the SA_RESETHAND flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will be delivered to or accepted by the process for every child process that terminates.
2. A single *wait()* issued from a SIGCHLD signal handler can be guaranteed to return immediately with status information for a child process.
3. When SA_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to receive a non-NULL pointer to a **siginfo_t** structure that describes a child process for which a wait via *waitpid()* or *waitid()* will not block or fail.
4. The *system()* function will not cause a process' SIGCHLD handler to be called as a result of the *fork()/exec* executed within *system()* because *system()* will accept the SIGCHLD signal when it performs a *waitpid()* for its child process. This is a desirable behavior of *system()* so that it can be used in a library without causing side effects to the application linked with the library.

An implementation that does not permit the *wait()* family of functions to accept (discard) a pending SIGCHLD signal associated with a successfully waited-for child, cannot make the guarantees described above for the following reasons:

Guarantee #1

Although it might be assumed that reliable queuing of all SIGCHLD signals generated by the system can make this guarantee, the counter-example is the case of a process that blocks SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the

49148 implementation supports queued signals, then eventually the system will run out of
 49149 memory for the queue. The guarantee cannot be made because there must be some limit to
 49150 the depth of queuing.

49151 Guarantees #2 and #3

49152 These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD
 49153 signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()*
 49154 function) will result in an invocation of the handler when SIGCHLD is unblocked, after the
 49155 process has disappeared.

49156 Guarantee #4

49157 Although possible to make this guarantee, *system()* would have to set the SIGCHLD
 49158 handler to SIG_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded
 49159 (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This
 49160 would have the undesirable side effect of discarding all SIGCHLD signals pending to the
 49161 process.

49162 FUTURE DIRECTIONS

49163 None.

49164 SEE ALSO

49165 *exec*, *exit()*, *fork()*, *waitid()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**,
 49166 **<sys/wait.h>**

49167 CHANGE HISTORY

49168 First released in Issue 1. Derived from Issue 1 of the SVID.

49169 Issue 5

49170 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

49171 Issue 6

49172 The following new requirements on POSIX implementations derive from alignment with the
 49173 Single UNIX Specification:

- 49174 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
 49175 required for conforming implementations of previous POSIX specifications, it was not
 49176 required for UNIX applications.

49177 The following changes were made to align with the IEEE P1003.1a draft standard:

- 49178 • The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

49179 The semantics of WIFSTOPPED(*stat_val*), WIFEXITED(*stat_val*), and WIFSIGNALED(*stat_val*)
 49180 are defined with respect to *posix_spawn()* or *posix_spawnnp()* for alignment with
 49181 IEEE Std 1003.1d-1999.

49182 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

49183 **NAME**

49184 waitid — wait for a child process to change state

49185 **SYNOPSIS**

49186 xSI #include <sys/wait.h>

49187 int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);

49188

49189 **DESCRIPTION**

49190 The *waitid()* function shall suspend the calling thread until one child of the process containing
 49191 the calling thread changes state. It records the current state of a child in the structure pointed to
 49192 by *infop*. If a child process changed state prior to the call to *waitid()*, *waitid()* shall return
 49193 immediately. If more than one thread is suspended in *wait()* or *waitpid()* waiting for termination
 49194 of the same process, exactly one thread shall return the process status at the time of the target
 49195 process termination.

49196 The *idtype* and *id* arguments are used to specify which children *waitid()* waits for.

49197 If *idtype* is P_PID, *waitid()* shall wait for the child with a process ID equal to (**pid_t**)*id*.

49198 If *idtype* is P_PGID, *waitid()* shall wait for any child with a process group ID equal to (**pid_t**)*id*.

49199 If *idtype* is P_ALL, *waitid()* shall wait for any children and *id* is ignored.

49200 The *options* argument is used to specify which state changes *waitid()* shall wait for. It is formed
 49201 by OR'ing together one or more of the following flags:

49202 WEXITED Wait for processes that have exited.

49203 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal.

49204 WCONTINUED Status shall be returned for any child that was stopped and has been
 49205 continued.

49206 WNOHANG Return immediately if there are no children to wait for.

49207 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This
 49208 shall not affect the state of the process; the process may be waited for again
 49209 after this call completes.

49210 The application shall ensure that the *infop* argument points to a **siginfo_t** structure. If *waitid()*
 49211 returns because a child process was found that satisfied the conditions indicated by the
 49212 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the
 49213 system with the status of the process. The *si_signo* member shall always be equal to SIGCHLD.

49214 **RETURN VALUE**

49215 If WNOHANG was specified and there are no children to wait for, 0 shall be returned. If *waitid()*
 49216 returns due to the change of state of one of its children, 0 shall be returned. Otherwise, -1 shall
 49217 be returned and *errno* set to indicate the error.

49218 **ERRORS**

49219 The *waitid()* function shall fail if:

49220 [ECHILD] The calling process has no existing unwaited-for child processes.

49221 [EINTR] The *waitid()* function was interrupted by a signal.

49222 [EINVAL] An invalid value was specified for *options*, or *idtype* and *id* specify an invalid
 49223 set of processes.

49224 **EXAMPLES**

49225 None.

49226 **APPLICATION USAGE**

49227 None.

49228 **RATIONALE**

49229 None.

49230 **FUTURE DIRECTIONS**

49231 None.

49232 **SEE ALSO**49233 *exec*, *exit()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/wait.h**>49234 **CHANGE HISTORY**

49235 First released in Issue 4, Version 2.

49236 **Issue 5**

49237 Moved from X/OPEN UNIX extension to BASE.

49238 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

49239 **Issue 6**

49240 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49241 **NAME**

49242 waitpid — wait for a child process to stop or terminate

49243 **SYNOPSIS**

49244 #include <sys/wait.h>

49245 pid_t waitpid(pid_t *pid*, int **stat_loc*, int *options*);49246 **DESCRIPTION**49247 Refer to *wait()*.

49248 **NAME**

49249 wrtomb — convert a wide-character code to a character (restartable)

49250 **SYNOPSIS**

49251 #include <stdio.h>

49252 size_t wrtomb(char *restrict *s*, wchar_t *wc*, mbstate_t *restrict *ps*);49253 **DESCRIPTION**

49254 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 49255 conflict between the requirements described here and the ISO C standard is unintentional. This
 49256 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49257 If *s* is a null pointer, the *wrtomb()* function shall be equivalent to the call:49258 wrtomb(*buf*, L'\0', *ps*)49259 where *buf* is an internal buffer.

49260 If *s* is not a null pointer, the *wrtomb()* function shall determine the number of bytes needed to
 49261 represent the character that corresponds to the wide character given by *wc* (including any shift
 49262 sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At
 49263 most {MB_CUR_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored,
 49264 preceded by any shift sequence needed to restore the initial shift state. The resulting state
 49265 described shall be the initial conversion state.

49266 If *ps* is a null pointer, the *wrtomb()* function shall use its own internal **mbstate_t** object, which is
 49267 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
 49268 pointed to by *ps* shall be used to completely describe the current conversion state of the
 49269 associated character sequence. The implementation shall behave as if no function defined in this
 49270 volume of IEEE Std 1003.1-2001 calls *wrtomb()*.

49271 CX If the application uses any of the _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS
 49272 functions, the application shall ensure that the *wrtomb()* function is called with a non-NULL *ps*
 49273 argument.

49274 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.49275 **RETURN VALUE**

49276 The *wrtomb()* function shall return the number of bytes stored in the array object (including any
 49277 shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this
 49278 case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)−1;
 49279 the conversion state shall be undefined.

49280 **ERRORS**49281 The *wrtomb()* function may fail if:49282 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

49283 [EILSEQ] Invalid wide-character code is detected.

49284 **EXAMPLES**

49285 None.

49286 **APPLICATION USAGE**

49287 None.

49288 **RATIONALE**

49289 None.

49290 **FUTURE DIRECTIONS**

49291 None.

49292 **SEE ALSO**49293 *mb sinit()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49294 **CHANGE HISTORY**49295 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
49296 (E).49297 **Issue 6**

49298 In the DESCRIPTION, a note on using this function in a threaded application is added.

49299 Extensions beyond the ISO C standard are marked.

49300 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49301 The *wcr tomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49302 **NAME**

49303 wcscat — concatenate two wide-character strings

49304 **SYNOPSIS**

49305 #include <wchar.h>

49306 wchar_t *wcscat(wchar_t *restrict *ws1*, const wchar_t *restrict *ws2*);49307 **DESCRIPTION**

49308 cx The functionality described on this reference page is aligned with the ISO C standard. Any
49309 conflict between the requirements described here and the ISO C standard is unintentional. This
49310 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49311 The *wcscat()* function shall append a copy of the wide-character string pointed to by *ws2*
49312 (including the terminating null wide-character code) to the end of the wide-character string
49313 pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character
49314 code at the end of *ws1*. If copying takes place between objects that overlap, the behavior is
49315 undefined.

49316 **RETURN VALUE**49317 The *wcscat()* function shall return *ws1*; no return value is reserved to indicate an error.49318 **ERRORS**

49319 No errors are defined.

49320 **EXAMPLES**

49321 None.

49322 **APPLICATION USAGE**

49323 None.

49324 **RATIONALE**

49325 None.

49326 **FUTURE DIRECTIONS**

49327 None.

49328 **SEE ALSO**49329 *wcncat()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49330 **CHANGE HISTORY**

49331 First released in Issue 4. Derived from the MSE working draft.

49332 **Issue 6**

49333 The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, *s1* is changed
49334 to *ws1*.

49335 The *wcscat()* prototype is updated for alignment with the ISO / IEC 9899:1999 standard.

49336 **NAME**

49337 wcschr — wide-character string scanning operation

49338 **SYNOPSIS**

49339 #include <wchar.h>

49340 wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

49341 **DESCRIPTION**

49342 cx The functionality described on this reference page is aligned with the ISO C standard. Any
49343 conflict between the requirements described here and the ISO C standard is unintentional. This
49344 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49345 The *wcschr()* function shall locate the first occurrence of *wc* in the wide-character string pointed
49346 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type
49347 **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The
49348 terminating null wide-character code is considered to be part of the wide-character string.

49349 **RETURN VALUE**

49350 Upon completion, *wcschr()* shall return a pointer to the wide-character code, or a null pointer if
49351 the wide-character code is not found.

49352 **ERRORS**

49353 No errors are defined.

49354 **EXAMPLES**

49355 None.

49356 **APPLICATION USAGE**

49357 None.

49358 **RATIONALE**

49359 None.

49360 **FUTURE DIRECTIONS**

49361 None.

49362 **SEE ALSO**49363 *wcsrchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49364 **CHANGE HISTORY**

49365 First released in Issue 4. Derived from the MSE working draft.

49366 **Issue 6**

49367 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49368 **NAME**

49369 wcscmp — compare two wide-character strings

49370 **SYNOPSIS**

49371 #include <wchar.h>

49372 int wcscmp(const wchar_t *ws1, const wchar_t *ws2);

49373 **DESCRIPTION**

49374 CX The functionality described on this reference page is aligned with the ISO C standard. Any
49375 conflict between the requirements described here and the ISO C standard is unintentional. This
49376 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49377 The *wcscmp()* function shall compare the wide-character string pointed to by *ws1* to the wide-
49378 character string pointed to by *ws2*.

49379 The sign of a non-zero return value shall be determined by the sign of the difference between the
49380 values of the first pair of wide-character codes that differ in the objects being compared.

49381 **RETURN VALUE**

49382 Upon completion, *wcscmp()* shall return an integer greater than, equal to, or less than 0, if the
49383 wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character
49384 string pointed to by *ws2*, respectively.

49385 **ERRORS**

49386 No errors are defined.

49387 **EXAMPLES**

49388 None.

49389 **APPLICATION USAGE**

49390 None.

49391 **RATIONALE**

49392 None.

49393 **FUTURE DIRECTIONS**

49394 None.

49395 **SEE ALSO**49396 *wcsncmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49397 **CHANGE HISTORY**

49398 First released in Issue 4. Derived from the MSE working draft.

49399 **NAME**

49400 wcscoll — wide-character string comparison using collating information

49401 **SYNOPSIS**

49402 #include <wchar.h>

49403 int wcscoll(const wchar_t *ws1, const wchar_t *ws2);

49404 **DESCRIPTION**

49405 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 49406 conflict between the requirements described here and the ISO C standard is unintentional. This
 49407 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49408 The *wcscoll()* function shall compare the wide-character string pointed to by *ws1* to the wide-
 49409 character string pointed to by *ws2*, both interpreted as appropriate to the *LC_COLLATE* category
 49410 of the current locale.

49411 CX The *wcscoll()* function shall not change the setting of *errno* if successful.

49412 An application wishing to check for error situations should set *errno* to 0 before calling *wcscoll()*.
 49413 If *errno* is non-zero on return, an error has occurred.

49414 **RETURN VALUE**

49415 Upon successful completion, *wcscoll()* shall return an integer greater than, equal to, or less than
 49416 0, according to whether the wide-character string pointed to by *ws1* is greater than, equal to, or
 49417 less than the wide-character string pointed to by *ws2*, when both are interpreted as appropriate
 49418 CX to the current locale. On error, *wcscoll()* shall set *errno*, but no return value is reserved to
 49419 indicate an error.

49420 **ERRORS**49421 The *wcscoll()* function may fail if:

49422 CX [EINVAL] The *ws1* or *ws2* arguments contain wide-character codes outside the domain of
 49423 the collating sequence.

49424 **EXAMPLES**

49425 None.

49426 **APPLICATION USAGE**49427 The *wcsxfrm()* and *wcscmp()* functions should be used for sorting large lists.49428 **RATIONALE**

49429 None.

49430 **FUTURE DIRECTIONS**

49431 None.

49432 **SEE ALSO**49433 *wcscmp()*, *wcsxfrm()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49434 **CHANGE HISTORY**

49435 First released in Issue 4. Derived from the MSE working draft.

49436 **Issue 5**

49437 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

49438 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

49439 **NAME**

49440 wcscpy — copy a wide-character string

49441 **SYNOPSIS**

49442 #include <wchar.h>

49443 wchar_t *wcscpy(wchar_t *restrict *ws1*, const wchar_t *restrict *ws2*);49444 **DESCRIPTION**

49445 cx The functionality described on this reference page is aligned with the ISO C standard. Any
49446 conflict between the requirements described here and the ISO C standard is unintentional. This
49447 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49448 The *wcscpy()* function shall copy the wide-character string pointed to by *ws2* (including the
49449 terminating null wide-character code) into the array pointed to by *ws1*. If copying takes place
49450 between objects that overlap, the behavior is undefined.

49451 **RETURN VALUE**49452 The *wcscpy()* function shall return *ws1*; no return value is reserved to indicate an error.49453 **ERRORS**

49454 No errors are defined.

49455 **EXAMPLES**

49456 None.

49457 **APPLICATION USAGE**

49458 None.

49459 **RATIONALE**

49460 None.

49461 **FUTURE DIRECTIONS**

49462 None.

49463 **SEE ALSO**49464 *wcsncpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49465 **CHANGE HISTORY**

49466 First released in Issue 4. Derived from the MSE working draft.

49467 **Issue 6**49468 The *wcscpy()* prototype is updated for alignment with the ISO / IEC 9899:1999 standard.

49469 **NAME**

49470 wcscspn — get the length of a complementary wide substring

49471 **SYNOPSIS**

49472 #include <wchar.h>

49473 size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);

49474 **DESCRIPTION**

49475 cx The functionality described on this reference page is aligned with the ISO C standard. Any
49476 conflict between the requirements described here and the ISO C standard is unintentional. This
49477 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49478 The *wcscspn()* function shall compute the length (in wide characters) of the maximum initial
49479 segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character
49480 codes *not* from the wide-character string pointed to by *ws2*.

49481 **RETURN VALUE**

49482 The *wcscspn()* function shall return the length of the initial substring of *ws1*; no return value is
49483 reserved to indicate an error.

49484 **ERRORS**

49485 No errors are defined.

49486 **EXAMPLES**

49487 None.

49488 **APPLICATION USAGE**

49489 None.

49490 **RATIONALE**

49491 None.

49492 **FUTURE DIRECTIONS**

49493 None.

49494 **SEE ALSO**49495 *wcssp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49496 **CHANGE HISTORY**

49497 First released in Issue 4. Derived from the MSE working draft.

49498 **Issue 5**

49499 The RETURN VALUE section is updated to indicate that *wcscspn()* returns the length of *ws1*,
49500 rather than *ws1* itself.

49501 **NAME**

49502 wcsftime — convert date and time to a wide-character string

49503 **SYNOPSIS**

49504 #include <wchar.h>

```
49505       size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,
49506                      const wchar_t *restrict format, const struct tm *restrict timeptr);
```

49507 **DESCRIPTION**

49508 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 49509 conflict between the requirements described here and the ISO C standard is unintentional. This
 49510 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49511 The *wcsftime()* function shall be equivalent to the *strftime()* function, except that:

- 49512 • The argument *wcs* points to the initial element of an array of wide characters into which the
 49513 generated output is to be placed.
- 49514 • The argument *maxsize* indicates the maximum number of wide characters to be placed in the
 49515 output array.
- 49516 • The argument *format* is a wide-character string and the conversion specifications are replaced
 49517 by corresponding sequences of wide characters.
- 49518 • The return value indicates the number of wide characters placed in the output array.

49519 If copying takes place between objects that overlap, the behavior is undefined.

49520 **RETURN VALUE**

49521 If the total number of resulting wide-character codes including the terminating null wide-
 49522 character code is no more than *maxsize*, *wcsftime()* shall return the number of wide-character
 49523 codes placed into the array pointed to by *wcs*, not including the terminating null wide-character
 49524 code. Otherwise, zero is returned and the contents of the array are unspecified.

49525 **ERRORS**

49526 No errors are defined.

49527 **EXAMPLES**

49528 None.

49529 **APPLICATION USAGE**

49530 None.

49531 **RATIONALE**

49532 None.

49533 **FUTURE DIRECTIONS**

49534 None.

49535 **SEE ALSO**

49536 *strftime()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>

49537 **CHANGE HISTORY**

49538 First released in Issue 4.

49539 **Issue 5**

49540 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

49541 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the *format*
 49542 argument is changed from **const char *** to **const wchar_t ***.

49543 **Issue 6**

49544 The *wcsftime()* prototype is updated for alignment with the ISO / IEC 9899: 1999 standard.

49545 **NAME**

49546 wcslen — get wide-character string length

49547 **SYNOPSIS**

49548 #include <wchar.h>

49549 size_t wcslen(const wchar_t *ws);

49550 **DESCRIPTION**

49551 cx The functionality described on this reference page is aligned with the ISO C standard. Any
49552 conflict between the requirements described here and the ISO C standard is unintentional. This
49553 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49554 The *wcslen()* function shall compute the number of wide-character codes in the wide-character
49555 string to which *ws* points, not including the terminating null wide-character code.

49556 **RETURN VALUE**

49557 The *wcslen()* function shall return the length of *ws*; no return value is reserved to indicate an
49558 error.

49559 **ERRORS**

49560 No errors are defined.

49561 **EXAMPLES**

49562 None.

49563 **APPLICATION USAGE**

49564 None.

49565 **RATIONALE**

49566 None.

49567 **FUTURE DIRECTIONS**

49568 None.

49569 **SEE ALSO**49570 The Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49571 **CHANGE HISTORY**

49572 First released in Issue 4. Derived from the MSE working draft.

49573 **NAME**49574 `wcsncat` — concatenate a wide-character string with part of another49575 **SYNOPSIS**49576 `#include <wchar.h>`49577 `wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,`
49578 `size_t n);`49579 **DESCRIPTION**49580 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
49581 conflict between the requirements described here and the ISO C standard is unintentional. This
49582 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.49583 The `wcsncat()` function shall append not more than *n* wide-character codes (a null wide-
49584 character code and wide-character codes that follow it are not appended) from the array pointed
49585 to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character
49586 code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null
49587 wide-character code shall always be appended to the result. If copying takes place between
49588 objects that overlap, the behavior is undefined.49589 **RETURN VALUE**49590 The `wcsncat()` function shall return *ws1*; no return value is reserved to indicate an error.49591 **ERRORS**

49592 No errors are defined.

49593 **EXAMPLES**

49594 None.

49595 **APPLICATION USAGE**

49596 None.

49597 **RATIONALE**

49598 None.

49599 **FUTURE DIRECTIONS**

49600 None.

49601 **SEE ALSO**49602 `wscat()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`49603 **CHANGE HISTORY**

49604 First released in Issue 4. Derived from the MSE working draft.

49605 **Issue 6**49606 The `wcsncat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49607 **NAME**

49608 wcsncmp — compare part of two wide-character strings

49609 **SYNOPSIS**

49610 #include <wchar.h>

49611 int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);

49612 **DESCRIPTION**

49613 CX The functionality described on this reference page is aligned with the ISO C standard. Any
49614 conflict between the requirements described here and the ISO C standard is unintentional. This
49615 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49616 The *wcsncmp()* function shall compare not more than *n* wide-character codes (wide-character
49617 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*
49618 to the array pointed to by *ws2*.

49619 The sign of a non-zero return value shall be determined by the sign of the difference between the
49620 values of the first pair of wide-character codes that differ in the objects being compared.

49621 **RETURN VALUE**

49622 Upon successful completion, *wcsncmp()* shall return an integer greater than, equal to, or less
49623 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less
49624 than the possibly null-terminated array pointed to by *ws2*, respectively.

49625 **ERRORS**

49626 No errors are defined.

49627 **EXAMPLES**

49628 None.

49629 **APPLICATION USAGE**

49630 None.

49631 **RATIONALE**

49632 None.

49633 **FUTURE DIRECTIONS**

49634 None.

49635 **SEE ALSO**49636 *wscmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49637 **CHANGE HISTORY**

49638 First released in Issue 4. Derived from the MSE working draft.

49639 **NAME**

49640 wcsncpy — copy part of a wide-character string

49641 **SYNOPSIS**

49642 #include <wchar.h>

49643 wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
49644 size_t n);49645 **DESCRIPTION**49646 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
49647 conflict between the requirements described here and the ISO C standard is unintentional. This
49648 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.49649 The *wcsncpy()* function shall copy not more than *n* wide-character codes (wide-character codes
49650 that follow a null wide-character code are not copied) from the array pointed to by *ws2* to the
49651 array pointed to by *ws1*. If copying takes place between objects that overlap, the behavior is
49652 undefined.49653 If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character
49654 codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*,
49655 until *n* wide-character codes in all are written.49656 **RETURN VALUE**49657 The *wcsncpy()* function shall return *ws1*; no return value is reserved to indicate an error.49658 **ERRORS**

49659 No errors are defined.

49660 **EXAMPLES**

49661 None.

49662 **APPLICATION USAGE**49663 If there is no null wide-character code in the first *n* wide-character codes of the array pointed to
49664 by *ws2*, the result is not null-terminated.49665 **RATIONALE**

49666 None.

49667 **FUTURE DIRECTIONS**

49668 None.

49669 **SEE ALSO**49670 *wscpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49671 **CHANGE HISTORY**

49672 First released in Issue 4. Derived from the MSE working draft.

49673 **Issue 6**49674 The *wcsncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49675 **NAME**

49676 wenspbrk — scan a wide-character string for a wide-character code

49677 **SYNOPSIS**

49678 #include <wchar.h>

49679 wchar_t *wenspbrk(const wchar_t *ws1, const wchar_t *ws2);

49680 **DESCRIPTION**

49681 cx The functionality described on this reference page is aligned with the ISO C standard. Any
49682 conflict between the requirements described here and the ISO C standard is unintentional. This
49683 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49684 The *wenspbrk()* function shall locate the first occurrence in the wide-character string pointed to by
49685 *ws1* of any wide-character code from the wide-character string pointed to by *ws2*.

49686 **RETURN VALUE**

49687 Upon successful completion, *wenspbrk()* shall return a pointer to the wide-character code or a null
49688 pointer if no wide-character code from *ws2* occurs in *ws1*.

49689 **ERRORS**

49690 No errors are defined.

49691 **EXAMPLES**

49692 None.

49693 **APPLICATION USAGE**

49694 None.

49695 **RATIONALE**

49696 None.

49697 **FUTURE DIRECTIONS**

49698 None.

49699 **SEE ALSO**49700 *wenschr()*, *wenschr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49701 **CHANGE HISTORY**

49702 First released in Issue 4. Derived from the MSE working draft.

49703 **NAME**

49704 wcsrchr — wide-character string scanning operation

49705 **SYNOPSIS**

49706 #include <wchar.h>

49707 wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

49708 **DESCRIPTION**

49709 cx The functionality described on this reference page is aligned with the ISO C standard. Any
49710 conflict between the requirements described here and the ISO C standard is unintentional. This
49711 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49712 The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed
49713 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type
49714 **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The
49715 terminating null wide-character code shall be considered to be part of the wide-character string.

49716 **RETURN VALUE**

49717 Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null
49718 pointer if *wc* does not occur in the wide-character string.

49719 **ERRORS**

49720 No errors are defined.

49721 **EXAMPLES**

49722 None.

49723 **APPLICATION USAGE**

49724 None.

49725 **RATIONALE**

49726 None.

49727 **FUTURE DIRECTIONS**

49728 None.

49729 **SEE ALSO**49730 *wcschr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49731 **CHANGE HISTORY**

49732 First released in Issue 4. Derived from the MSE working draft.

49733 **Issue 6**

49734 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49735 **NAME**

49736 wcsrtoombs — convert a wide-character string to a character string (restartable)

49737 **SYNOPSIS**

49738 #include <wchar.h>

```
49739       size_t wcsrtoombs(char *restrict dst, const wchar_t **restrict src,
49740                       size_t len, mbstate_t *restrict ps);
```

49741 **DESCRIPTION**

49742 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 49743 conflict between the requirements described here and the ISO C standard is unintentional. This
 49744 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49745 The *wcsrtoombs()* function shall convert a sequence of wide characters from the array indirectly
 49746 pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state
 49747 described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters
 49748 shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a
 49749 terminating null wide character, which shall also be stored. Conversion shall stop earlier in the
 49750 following cases:

- 49751 • When a code is reached that does not correspond to a valid character
- 49752 • When the next character would exceed the limit of *len* total bytes to be stored in the array
 49753 pointed to by *dst* (and *dst* is not a null pointer)

49754 Each conversion shall take place as if by a call to the *wcrtomb()* function.

49755 If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null
 49756 pointer (if conversion stopped due to reaching a terminating null wide character) or the address
 49757 just past the last wide character converted (if any). If conversion stopped due to reaching a
 49758 terminating null wide character, the resulting state described shall be the initial conversion state.

49759 If *ps* is a null pointer, the *wcsrtoombs()* function shall use its own internal **mbstate_t** object, which
 49760 is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
 49761 pointed to by *ps* shall be used to completely describe the current conversion state of the
 49762 associated character sequence. The implementation shall behave as if no function defined in this
 49763 volume of IEEE Std 1003.1-2001 calls *wcsrtoombs()*.

49764 CX If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`
 49765 functions, the application shall ensure that the *wcsrtoombs()* function is called with a non-NULL
 49766 *ps* argument.

49767 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

49768 **RETURN VALUE**

49769 If conversion stops because a code is reached that does not correspond to a valid character, an
 49770 encoding error occurs. In this case, the *wcsrtoombs()* function shall store the value of the macro
 49771 [EILSEQ] in *errno* and return (**size_t**)−1; the conversion state is undefined. Otherwise, it shall
 49772 return the number of bytes in the resulting character sequence, not including the terminating
 49773 null (if any).

49774 **ERRORS**

49775 The *wcsrtoombs()* function may fail if:

- 49776 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.
- 49777 [EILSEQ] A wide-character code does not correspond to a valid character.

49778 **EXAMPLES**

49779 None.

49780 **APPLICATION USAGE**

49781 None.

49782 **RATIONALE**

49783 None.

49784 **FUTURE DIRECTIONS**

49785 None.

49786 **SEE ALSO**49787 *mbstinit()*, *wcrtomb()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49788 **CHANGE HISTORY**49789 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
49790 (E).49791 **Issue 6**

49792 In the DESCRIPTION, a note on using this function in a threaded application is added.

49793 Extensions beyond the ISO C standard are marked.

49794 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49795 The *wcsrtombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49796 **NAME**

49797 wcssp — get the length of a wide substring

49798 **SYNOPSIS**

49799 #include <wchar.h>

49800 size_t wcssp(const wchar_t *ws1, const wchar_t *ws2);

49801 **DESCRIPTION**

49802 cx The functionality described on this reference page is aligned with the ISO C standard. Any
49803 conflict between the requirements described here and the ISO C standard is unintentional. This
49804 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49805 The wcssp() function shall compute the length (in wide characters) of the maximum initial
49806 segment of the wide-character string pointed to by ws1 which consists entirely of wide-character
49807 codes from the wide-character string pointed to by ws2.

49808 **RETURN VALUE**

49809 The wcssp() function shall return the length of the initial substring of ws1; no return value is
49810 reserved to indicate an error.

49811 **ERRORS**

49812 No errors are defined.

49813 **EXAMPLES**

49814 None.

49815 **APPLICATION USAGE**

49816 None.

49817 **RATIONALE**

49818 None.

49819 **FUTURE DIRECTIONS**

49820 None.

49821 **SEE ALSO**

49822 wcscsp(), the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>

49823 **CHANGE HISTORY**

49824 First released in Issue 4. Derived from the MSE working draft.

49825 **Issue 5**

49826 The RETURN VALUE section is updated to indicate that wcssp() returns the length of ws1
49827 rather than ws1 itself.

49828 **NAME**49829 `wcsstr` — find a wide-character substring49830 **SYNOPSIS**49831 `#include <wchar.h>`49832 `wchar_t *wcsstr(const wchar_t *restrict ws1,`
49833 `const wchar_t *restrict ws2);`49834 **DESCRIPTION**49835 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
49836 conflict between the requirements described here and the ISO C standard is unintentional. This
49837 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.49838 The `wcsstr()` function shall locate the first occurrence in the wide-character string pointed to by
49839 `ws1` of the sequence of wide characters (excluding the terminating null wide character) in the
49840 wide-character string pointed to by `ws2`.49841 **RETURN VALUE**49842 Upon successful completion, `wcsstr()` shall return a pointer to the located wide-character string,
49843 or a null pointer if the wide-character string is not found.49844 If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.49845 **ERRORS**

49846 No errors are defined.

49847 **EXAMPLES**

49848 None.

49849 **APPLICATION USAGE**

49850 None.

49851 **RATIONALE**

49852 None.

49853 **FUTURE DIRECTIONS**

49854 None.

49855 **SEE ALSO**49856 `wcschr()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`49857 **CHANGE HISTORY**49858 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
49859 (E).49860 **Issue 6**49861 The `wcsstr()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49862 **NAME**

49863 wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

49864 **SYNOPSIS**

49865 #include <wchar.h>

49866 double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);

49867 float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);

49868 long double wcstold(const wchar_t *restrict nptr,

49869 wchar_t **restrict endptr);

49870 **DESCRIPTION**

49871 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 49872 conflict between the requirements described here and the ISO C standard is unintentional. This
 49873 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49874 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to
 49875 **double**, **float**, and **long double** representation, respectively. First, they shall decompose the
 49876 input wide-character string into three parts:

- 49877 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
 49878 *iswspace()*)
- 49879 2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
- 49880 3. A final wide-character string of one or more unrecognized wide-character codes, including
 49881 the terminating null wide-character code of the input wide-character string

49882 Then they shall attempt to convert the subject sequence to a floating-point number, and return
 49883 the result.

49884 The expected form of the subject sequence is an optional plus or minus sign, then one of the
 49885 following:

- 49886 • A non-empty sequence of decimal digits optionally containing a radix character, then an
 49887 optional exponent part
- 49888 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix
 49889 character, then an optional binary exponent part
- 49890 • One of INF or INFINITY, or any other wide string equivalent except for case
- 49891 • One of NAN or NAN(*n-wchar-sequence_{opt}*), or any other wide string ignoring case in the NAN
 49892 part, where:

49893 n-wchar-sequence:

49894 digit

49895 nondigit

49896 n-wchar-sequence digit

49897 n-wchar-sequence nondigit

49898 The subject sequence is defined as the longest initial subsequence of the input wide string,
 49899 starting with the first non-white-space wide character, that is of the expected form. The subject
 49900 sequence contains no wide characters if the input wide string is not of the expected form.

49901 If the subject sequence has the expected form for a floating-point number, the sequence of wide
 49902 characters starting with the first digit or the radix character (whichever occurs first) shall be
 49903 interpreted as a floating constant according to the rules of the C language, except that the radix
 49904 character shall be used in place of a period, and that if neither an exponent part nor a radix
 49905 character appears in a decimal floating-point number, or if a binary exponent part does not

49906 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with
 49907 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins
 49908 with a minus sign, the sequence shall be interpreted as negated. A wide-character sequence INF
 49909 or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it
 49910 were a floating constant that is too large for the range of the return type. A wide-character
 49911 sequence NAN or NAN(*n-wchar-sequence_{opt}*) shall be interpreted as a quiet NaN, if supported in
 49912 the return type, else as if it were a subject sequence part that does not have the expected form;
 49913 the meaning of the *n-wchar* sequences is implementation-defined. A pointer to the final wide
 49914 string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

49915 If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the
 49916 conversion shall be rounded in an implementation-defined manner.

49917 CX The radix character shall be as defined in the program's locale (category *LC_NUMERIC*). In the
 49918 POSIX locale, or in a locale where the radix character is not defined, the radix character shall
 49919 default to a period ('.').

49920 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
 49921 accepted.

49922 If the subject sequence is empty or does not have the expected form, no conversion shall be
 49923 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
 49924 *endptr* is not a null pointer.

49925 CX The *wcstod()* function shall not change the setting of *errno* if successful.

49926 Since 0 is returned on error and is also a valid return on success, an application wishing to check
 49927 for error situations should set *errno* to 0, then call *wcstod()*, *wcstof()*, or *wcstold()*, then check
 49928 *errno*.

49929 RETURN VALUE

49930 Upon successful completion, these functions shall return the converted value. If no conversion
 49931 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

49932 If the correct value is outside the range of representable values, ±HUGE_VAL, ±HUGE_VALF, or |
 49933 ±HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to |
 49934 [ERANGE].

49935 If the correct value would cause underflow, a value whose magnitude is no greater than the
 49936 smallest normalized positive number in the return type shall be returned and *errno* set to
 49937 [ERANGE].

49938 ERRORS

49939 The *wcstod()* function shall fail if:

49940 [ERANGE] The value to be returned would cause overflow or underflow.

49941 The *wcstod()* function may fail if:

49942 CX [EINVAL] No conversion could be performed.

49943 **EXAMPLES**

49944 None.

49945 **APPLICATION USAGE**

49946 If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the
 49947 result is not exactly representable, the result should be one of the two numbers in the
 49948 appropriate internal format that are adjacent to the hexadecimal floating source value, with the
 49949 extra stipulation that the error should have a correct sign for the current rounding direction.

49950 If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in **<float.h>**)
 49951 significant digits, the result should be correctly rounded. If the subject sequence *D* has the
 49952 decimal form and more than DECIMAL_DIG significant digits, consider the two bounding,
 49953 adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the
 49954 values of *L*, *D*, and *U* satisfy "*L* ≤ *D* ≤ *U*". The result should be one of the (equal or
 49955 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current
 49956 rounding direction, with the extra stipulation that the error with respect to *D* should have a
 49957 correct sign for the current rounding direction.

49958 **RATIONALE**

49959 None.

49960 **FUTURE DIRECTIONS**

49961 None.

49962 **SEE ALSO**

49963 *iswspace()*, *localeconv()*, *scanf()*, *setlocale()*, *wcstol()*, the Base Definitions volume of
 49964 IEEE Std 1003.1-2001, Chapter 7, Locale, **<float.h>**, **<wchar.h>**

49965 **CHANGE HISTORY**

49966 First released in Issue 4. Derived from the MSE working draft.

49967 **Issue 5**49968 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.49969 **Issue 6**

49970 Extensions beyond the ISO C standard are marked.

49971 The following new requirements on POSIX implementations derive from alignment with the
 49972 Single UNIX Specification:

- 49973 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
- 49974 added if no conversion could be performed.

49975 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 49976 • The *wcstod()* prototype is updated.
- 49977 • The *wcstof()* and *wcstold()* functions are added.
- 49978 • If the correct value for *wcstod()* would cause underflow, the return value changed from 0 (as
- 49979 specified in Issue 5) to the smallest normalized positive number.
- 49980 • The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively
- 49981 updated.

49982 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

49983 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second
 49984 paragraph in the RETURN VALUE section.

49985 **NAME**

49986 wcstoimax, wcstoumax — convert a wide-character string to an integer type

49987 **SYNOPSIS**

49988 #include <stddef.h>

49989 #include <inttypes.h>

49990 intmax_t wcstoimax(const wchar_t *restrict nptr,

49991 wchar_t **restrict endptr, int base);

49992 uintmax_t wcstoumax(const wchar_t *restrict nptr,

49993 wchar_t **restrict endptr, int base);

49994 **DESCRIPTION**

49995 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 49996 conflict between the requirements described here and the ISO C standard is unintentional. This
 49997 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49998 These functions shall be equivalent to the *wcstol()*, *wcstoll()*, *wcstoul()*, and *wcstoull()* functions,
 49999 respectively, except that the initial portion of the wide string shall be converted to **intmax_t** and
 50000 **uintmax_t** representation, respectively.

50001 **RETURN VALUE**

50002 These functions shall return the converted value, if any.

50003 If no conversion could be performed, zero shall be returned. If the correct value is outside the
 50004 range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall
 50005 be returned (according to the return type and sign of the value, if any), and *errno* shall be set to
 50006 [ERANGE].

50007 **ERRORS**

50008 These functions shall fail if:

50009 [EINVAL] The value of *base* is not supported.

50010 [ERANGE] The value to be returned is not representable.

50011 These functions may fail if:

50012 [EINVAL] No conversion could be performed.

50013 **EXAMPLES**

50014 None.

50015 **APPLICATION USAGE**

50016 None.

50017 **RATIONALE**

50018 None.

50019 **FUTURE DIRECTIONS**

50020 None.

50021 **SEE ALSO**50022 *wcstol()*, *wcstoul()*, the Base Definitions volume of IEEE Std 1003.1-2001, <inttypes.h>,
 50023 <stddef.h>50024 **CHANGE HISTORY**

50025 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

50026 **NAME**

50027 wcstok — split a wide-character string into tokens

50028 **SYNOPSIS**

50029 #include <wchar.h>

50030 wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,
 50031 wchar_t **restrict ptr);

50032 **DESCRIPTION**

50033 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 50034 conflict between the requirements described here and the ISO C standard is unintentional. This
 50035 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50036 A sequence of calls to *wcstok()* shall break the wide-character string pointed to by *ws1* into a
 50037 sequence of tokens, each of which shall be delimited by a wide-character code from the wide-
 50038 character string pointed to by *ws2*. The *ptr* argument points to a caller-provided **wchar_t** pointer
 50039 into which the *wcstok()* function shall store information necessary for it to continue scanning the
 50040 same wide-character string.

50041 The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null
 50042 pointer as their first argument. The separator string pointed to by *ws2* may be different from call
 50043 to call.

50044 The first call in the sequence shall search the wide-character string pointed to by *ws1* for the first
 50045 wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no
 50046 such wide-character code is found, then there are no tokens in the wide-character string pointed
 50047 to by *ws1* and *wcstok()* shall return a null pointer. If such a wide-character code is found, it shall
 50048 be the start of the first token.

50049 The *wcstok()* function shall then search from there for a wide-character code that *is* contained in
 50050 the current separator string. If no such wide-character code is found, the current token extends
 50051 to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token
 50052 shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a
 50053 null wide character, which terminates the current token. The *wcstok()* function shall save a
 50054 pointer to the following wide-character code, from which the next search for a token shall start.

50055 Each subsequent call, with a null pointer as the value of the first argument, shall start searching
 50056 from the saved pointer and behave as described above.

50057 The implementation shall behave as if no function calls *wcstok()*.

50058 **RETURN VALUE**

50059 Upon successful completion, the *wcstok()* function shall return a pointer to the first wide-
 50060 character code of a token. Otherwise, if there is no token, *wcstok()* shall return a null pointer.

50061 **ERRORS**

50062 No errors are defined.

50063 **EXAMPLES**

50064 None.

50065 **APPLICATION USAGE**

50066 None.

50067 **RATIONALE**

50068 None.

50069 **FUTURE DIRECTIONS**

50070 None.

50071 **SEE ALSO**

50072 The Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>

50073 **CHANGE HISTORY**

50074 First released in Issue 4.

50075 **Issue 5**50076 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is
50077 added to the definition of *wcstok()* in the SYNOPSIS.50078 **Issue 6**50079 The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

50080 **NAME**

50081 wcstol, wcstoll — convert a wide-character string to a long integer

50082 **SYNOPSIS**

50083 #include <wchar.h>

50084 long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,
50085 int base);50086 long long wcstoll(const wchar_t *restrict nptr,
50087 wchar_t **restrict endptr, int base);50088 **DESCRIPTION**50089 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
50090 conflict between the requirements described here and the ISO C standard is unintentional. This
50091 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.50092 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to
50093 **long**, **long long**, **unsigned long**, and **unsigned long long** representation, respectively. First, they
50094 shall decompose the input string into three parts:

- 50095 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
50096 *iswspace()*)
- 50097 2. A subject sequence interpreted as an integer represented in some radix determined by the
50098 value of *base*
- 50099 3. A final wide-character string of one or more unrecognized wide-character codes, including
50100 the terminating null wide-character code of the input wide-character string

50101 Then they shall attempt to convert the subject sequence to an integer, and return the result.

50102 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,
50103 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal
50104 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal
50105 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'
50106 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
50107 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.50108 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
50109 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
50110 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'
50111 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less
50112 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code
50113 representations of 0x or 0X may optionally precede the sequence of letters and digits, following
50114 the sign if present.50115 The subject sequence is defined as the longest initial subsequence of the input wide-character
50116 string, starting with the first non-white-space wide-character code that is of the expected form.
50117 The subject sequence contains no wide-character codes if the input wide-character string is
50118 empty or consists entirely of white-space wide-character code, or if the first non-white-space
50119 wide-character code is other than a sign or a permissible letter or digit.50120 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes
50121 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has
50122 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for
50123 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a
50124 minus sign, the value resulting from the conversion shall be negated. A pointer to the final
50125 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is

- 50126 not a null pointer.
- 50127 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
50128 accepted.
- 50129 If the subject sequence is empty or does not have the expected form, no conversion shall be
50130 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
50131 *endptr* is not a null pointer.
- 50132 CX These functions shall not change the setting of *errno* if successful.
- 50133 Since 0, {LONG_MIN} or {LLONG_MIN} and {LONG_MAX} or {LLONG_MAX} are returned on
50134 error and are also valid returns on success, an application wishing to check for error situations
50135 should set *errno* to 0, then call *wcstol()* or *wcstoll()*, then check *errno*.
- 50136 **RETURN VALUE**
- 50137 Upon successful completion, these functions shall return the converted value, if any. If no
50138 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If
50139 the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
50140 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
50141 *errno* set to [ERANGE].
- 50142 **ERRORS**
- 50143 These functions shall fail if:
- 50144 CX [EINVAL] The value of *base* is not supported.
- 50145 [ERANGE] The value to be returned is not representable.
- 50146 These functions may fail if:
- 50147 CX [EINVAL] No conversion could be performed.
- 50148 **EXAMPLES**
- 50149 None.
- 50150 **APPLICATION USAGE**
- 50151 None.
- 50152 **RATIONALE**
- 50153 None.
- 50154 **FUTURE DIRECTIONS**
- 50155 None.
- 50156 **SEE ALSO**
- 50157 *iswalpha()*, *scanf()*, *wcstod()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>
- 50158 **CHANGE HISTORY**
- 50159 First released in Issue 4. Derived from the MSE working draft.
- 50160 **Issue 5**
- 50161 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 50162 **Issue 6**
- 50163 Extensions beyond the ISO C standard are marked.
- 50164 The following new requirements on POSIX implementations derive from alignment with the
50165 Single UNIX Specification:
- 50166 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
50167 added if no conversion could be performed.

50168 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

50169 • The *wcstol()* prototype is updated.

50170 • The *wcstoll()* function is added.

50171 **NAME**

50172 wcstold — convert a wide-character string to a double-precision number

50173 **SYNOPSIS**

50174 #include <wchar.h>

50175 long double wcstold(const wchar_t *restrict *nptr*,

50176 wchar_t **restrict *endptr*);

50177 **DESCRIPTION**

50178 Refer to *wcstod()*.

50179 **NAME**

50180 wcstoll — convert a wide-character string to a long integer

50181 **SYNOPSIS**

50182 #include <wchar.h>

50183 long long wcstoll(const wchar_t *restrict *nptr*,50184 wchar_t **restrict *endptr*, int *base*);50185 **DESCRIPTION**50186 Refer to *wcstol()*.

50187 **NAME**

50188 wcstombs — convert a wide-character string to a character string

50189 **SYNOPSIS**

50190 #include <stdlib.h>

50191 size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,
50192 size_t n);

50193 **DESCRIPTION**

50194 CX The functionality described on this reference page is aligned with the ISO C standard. Any
50195 conflict between the requirements described here and the ISO C standard is unintentional. This
50196 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50197 The *wcstombs()* function shall convert the sequence of wide-character codes that are in the array
50198 pointed to by *pwcs* into a sequence of characters that begins in the initial shift state and store
50199 these characters into the array pointed to by *s*, stopping if a character would exceed the limit of *n*
50200 total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to
50201 *wctomb()*, except that the shift state of *wctomb()* shall not be affected.

50202 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

50203 No more than *n* bytes shall be modified in the array pointed to by *s*. If copying takes place
50204 CX between objects that overlap, the behavior is undefined. If *s* is a null pointer, *wcstombs()* shall
50205 return the length required to convert the entire array regardless of the value of *n*, but no values
50206 are stored.

50207 The *wcstombs()* function need not be reentrant. A function that is not required to be reentrant is
50208 not required to be thread-safe.

50209 **RETURN VALUE**

50210 If a wide-character code is encountered that does not correspond to a valid character (of one or
50211 more bytes each), *wcstombs()* shall return **(size_t)−1**. Otherwise, *wcstombs()* shall return the
50212 number of bytes stored in the character array, not including any terminating null byte. The array
50213 shall not be null-terminated if the value returned is *n*.

50214 **ERRORS**

50215 The *wcstombs()* function may fail if:

50216 CX [EILSEQ] A wide-character code does not correspond to a valid character.

50217 **EXAMPLES**

50218 None.

50219 **APPLICATION USAGE**

50220 None.

50221 **RATIONALE**

50222 None.

50223 **FUTURE DIRECTIONS**

50224 None.

50225 **SEE ALSO**

50226 *mblen()*, *mbtowc()*, *mbstowcs()*, *wctomb()*, the Base Definitions volume of IEEE Std 1003.1-2001,
50227 <stdlib.h>

50228 **CHANGE HISTORY**

50229 First released in Issue 4. Derived from the ISO C standard.

50230 **Issue 6**

50231 The following new requirements on POSIX implementations derive from alignment with the
50232 Single UNIX Specification:

- 50233 • The DESCRIPTION states the effect of when *s* is a null pointer.
- 50234 • The [EILSEQ] error condition is added.

50235 The *wcstombs()* prototype is updated for alignment with the ISO / IEC 9899:1999 standard.

50236 **NAME**

50237 wcstoul, wcstoull — convert a wide-character string to an unsigned long

50238 **SYNOPSIS**

50239 #include <wchar.h>

50240 unsigned long wcstoul(const wchar_t *restrict nptr,

50241 wchar_t **restrict endptr, int base);

50242 unsigned long long wcstoull(const wchar_t *restrict nptr,

50243 wchar_t **restrict endptr, int base);

50244 **DESCRIPTION**

50245 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 50246 conflict between the requirements described here and the ISO C standard is unintentional. This
 50247 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50248 The `wcstoul()` and `wcstoull()` functions shall convert the initial portion of the wide-character
 50249 string pointed to by *nptr* to **unsigned long** and **unsigned long long** representation, respectively.
 50250 First, they shall decompose the input wide-character string into three parts:

- 50251 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
 50252 *iswspace()*)
- 50253 2. A subject sequence interpreted as an integer represented in some radix determined by the
 50254 value of *base*
- 50255 3. A final wide-character string of one or more unrecognized wide-character codes, including
 50256 the terminating null wide-character code of the input wide-character string

50257 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the
 50258 result.

50259 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,
 50260 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal
 50261 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal
 50262 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'
 50263 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
 50264 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

50265 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
 50266 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
 50267 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'
 50268 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less
 50269 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character codes 0x or 0X
 50270 may optionally precede the sequence of letters and digits, following the sign if present.

50271 The subject sequence is defined as the longest initial subsequence of the input wide-character
 50272 string, starting with the first wide-character code that is not white space and is of the expected
 50273 form. The subject sequence contains no wide-character codes if the input wide-character string is
 50274 empty or consists entirely of white-space wide-character codes, or if the first wide-character
 50275 code that is not white space is other than a sign or a permissible letter or digit.

50276 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes
 50277 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has
 50278 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for
 50279 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a
 50280 minus sign, the value resulting from the conversion shall be negated. A pointer to the final
 50281 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is

- 50282 not a null pointer.
- 50283 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
50284 accepted.
- 50285 If the subject sequence is empty or does not have the expected form, no conversion shall be
50286 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
50287 *endptr* is not a null pointer.
- 50288 CX The *wcstoul()* function shall not change the setting of *errno* if successful.
- 50289 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and 0 is also a valid return
50290 on success, an application wishing to check for error situations should set *errno* to 0, then call
50291 *wcstoul()* or *wcstoull()*, then check *errno*.
- 50292 **RETURN VALUE**
- 50293 Upon successful completion, the *wcstoul()* and *wcstoull()* functions shall return the converted
50294 CX value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to
50295 indicate the error. If the correct value is outside the range of representable values,
50296 {ULONG_MAX} or {ULLONG_MAX} respectively shall be returned and *errno* set to [ERANGE].
- 50297 **ERRORS**
- 50298 These functions shall fail if:
- 50299 CX [EINVAL] The value of *base* is not supported.
- 50300 [ERANGE] The value to be returned is not representable.
- 50301 These functions may fail if:
- 50302 CX [EINVAL] No conversion could be performed.
- 50303 **EXAMPLES**
- 50304 None.
- 50305 **APPLICATION USAGE**
- 50306 None.
- 50307 **RATIONALE**
- 50308 None.
- 50309 **FUTURE DIRECTIONS**
- 50310 None.
- 50311 **SEE ALSO**
- 50312 *iswalph()*, *scanf()*, *wcstod()*, *wcstol()*, the Base Definitions volume of IEEE Std 1003.1-2001,
50313 <wchar.h>
- 50314 **CHANGE HISTORY**
- 50315 First released in Issue 4. Derived from the MSE working draft.
- 50316 **Issue 5**
- 50317 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 50318 **Issue 6**
- 50319 Extensions beyond the ISO C standard are marked.
- 50320 The following new requirements on POSIX implementations derive from alignment with the
50321 Single UNIX Specification:
- 50322 • The [EINVAL] error condition is added for when the value of *base* is not supported.

- 50323 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
50324 added if no conversion could be performed.
- 50325 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 50326 • The *wcstoul()* prototype is updated.
- 50327 • The *wcstoull()* function is added.

50328 **NAME**

50329 wcstoumax — convert a wide-character string to an integer type

50330 **SYNOPSIS**

50331 #include <stddef.h>

50332 #include <inttypes.h>

50333 uintmax_t wcstoumax(const wchar_t *restrict nptr,

50334 wchar_t **restrict endptr, int base);

50335 **DESCRIPTION**50336 Refer to *wcstoimax()*.

50337 **NAME**50338 wcswcs — find a wide substring (**LEGACY**)50339 **SYNOPSIS**

50340 xSI #include <wchar.h>

50341 wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2);

50342

50343 **DESCRIPTION**

50344 The wcswcs() function shall locate the first occurrence in the wide-character string pointed to by
50345 ws1 of the sequence of wide-character codes (excluding the terminating null wide-character
50346 code) in the wide-character string pointed to by ws2.

50347 **RETURN VALUE**

50348 Upon successful completion, wcswcs() shall return a pointer to the located wide-character string
50349 or a null pointer if the wide-character string is not found.

50350 If ws2 points to a wide-character string with zero length, the function shall return ws1.

50351 **ERRORS**

50352 No errors are defined.

50353 **EXAMPLES**

50354 None.

50355 **APPLICATION USAGE**

50356 This function was not included in the final ISO/IEC 9899:1990/Amendment 1:1995 (E).
50357 Application developers are strongly encouraged to use the wcsstr() function instead.

50358 **RATIONALE**

50359 None.

50360 **FUTURE DIRECTIONS**

50361 This function may be withdrawn in a future version.

50362 **SEE ALSO**

50363 wcschr(), wcsstr(), the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>

50364 **CHANGE HISTORY**

50365 First released in Issue 4. Derived from the MSE working draft.

50366 **Issue 5**

50367 Marked EX.

50368 **Issue 6**

50369 This function is marked LEGACY.

50370 **NAME**

50371 wcswidth — number of column positions of a wide-character string

50372 **SYNOPSIS**

50373 xSI #include <wchar.h>

50374 int wcswidth(const wchar_t *pwcs, size_t n);

50375

50376 **DESCRIPTION**

50377 The *wcswidth()* function shall determine the number of column positions required for *n* wide-character codes (or fewer than *n* wide-character codes if a null wide-character code is encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

50380 **RETURN VALUE**

50381 The *wcswidth()* function either shall return 0 (if *pwcs* points to a null wide-character code), or return the number of column positions to be occupied by the wide-character string pointed to by *pwcs*, or return -1 (if any of the first *n* wide-character codes in the wide-character string pointed to by *pwcs* is not a printable wide-character code).

50385 **ERRORS**

50386 No errors are defined.

50387 **EXAMPLES**

50388 None.

50389 **APPLICATION USAGE**

50390 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the return value for a non-printable wide character is not specified.

50392 **RATIONALE**

50393 None.

50394 **FUTURE DIRECTIONS**

50395 None.

50396 **SEE ALSO**

50397 *wcwidth()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.103, Column Position, <wchar.h>

50399 **CHANGE HISTORY**

50400 First released in Issue 4. Derived from the MSE working draft.

50401 **Issue 6**

50402 The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

50403 **NAME**50404 `wcsxfrm` — wide-character string transformation50405 **SYNOPSIS**50406 `#include <wchar.h>`

```
50407        size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,  

50408                size_t n);
```

50409 **DESCRIPTION**

50410 CX The functionality described on this reference page is aligned with the ISO C standard. Any
50411 conflict between the requirements described here and the ISO C standard is unintentional. This
50412 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50413 The `wcsxfrm()` function shall transform the wide-character string pointed to by `ws2` and place the
50414 resulting wide-character string into the array pointed to by `ws1`. The transformation shall be
50415 such that if `wcscmp()` is applied to two transformed wide strings, it shall return a value greater
50416 than, equal to, or less than 0, corresponding to the result of `wscoll()` applied to the same two
50417 original wide-character strings. No more than *n* wide-character codes shall be placed into the
50418 resulting array pointed to by `ws1`, including the terminating null wide-character code. If *n* is 0,
50419 `ws1` is permitted to be a null pointer. If copying takes place between objects that overlap, the
50420 behavior is undefined.

50421 CX The `wcsxfrm()` function shall not change the setting of `errno` if successful.

50422 Since no return value is reserved to indicate an error, an application wishing to check for error
50423 situations should set `errno` to 0, then call `wcsxfrm()`, then check `errno`.

50424 **RETURN VALUE**

50425 The `wcsxfrm()` function shall return the length of the transformed wide-character string (not
50426 including the terminating null wide-character code). If the value returned is *n* or more, the
50427 contents of the array pointed to by `ws1` are unspecified.

50428 CX On error, the `wcsxfrm()` function may set `errno`, but no return value is reserved to indicate an
50429 error.

50430 **ERRORS**

50431 The `wcsxfrm()` function may fail if:

50432 CX [EINVAL] The wide-character string pointed to by `ws2` contains wide-character codes
50433 outside the domain of the collating sequence.

50434 **EXAMPLES**

50435 None.

50436 **APPLICATION USAGE**

50437 The transformation function is such that two transformed wide-character strings can be ordered
50438 by `wcscmp()` as appropriate to collating sequence information in the program's locale (category
50439 `LC_COLLATE`).

50440 The fact that when *n* is 0 `ws1` is permitted to be a null pointer is useful to determine the size of
50441 the `ws1` array prior to making the transformation.

50442 **RATIONALE**

50443 None.

50444 **FUTURE DIRECTIONS**

50445 None.

50446 **SEE ALSO**50447 `wscmp()`, `wscoll()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`50448 **CHANGE HISTORY**

50449 First released in Issue 4. Derived from the MSE working draft.

50450 **Issue 5**

50451 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

50452 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.50453 **Issue 6**50454 In previous versions, this function was required to return `-1` on error.

50455 Extensions beyond the ISO C standard are marked.

50456 The following new requirements on POSIX implementations derive from alignment with the
50457 Single UNIX Specification:

- 50458
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
-
- 50459 added if no conversion could be performed.

50460 The `wcsxfrm()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

50461 **NAME**

50462 wctob — wide-character to single-byte conversion

50463 **SYNOPSIS**

50464 #include <stdio.h>

50465 #include <wchar.h>

50466 int wctob(wint_t c);

50467 **DESCRIPTION**

50468 cx The functionality described on this reference page is aligned with the ISO C standard. Any
50469 conflict between the requirements described here and the ISO C standard is unintentional. This
50470 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50471 The *wctob()* function shall determine whether *c* corresponds to a member of the extended
50472 character set whose character representation is a single byte when in the initial shift state.

50473 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

50474 **RETURN VALUE**

50475 The *wctob()* function shall return EOF if *c* does not correspond to a character with length one in
50476 the initial shift state. Otherwise, it shall return the single-byte representation of that character as
50477 an **unsigned char** converted to **int**.

50478 **ERRORS**

50479 No errors are defined.

50480 **EXAMPLES**

50481 None.

50482 **APPLICATION USAGE**

50483 None.

50484 **RATIONALE**

50485 None.

50486 **FUTURE DIRECTIONS**

50487 None.

50488 **SEE ALSO**50489 *btowc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>50490 **CHANGE HISTORY**

50491 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
50492 (E).

50493 **NAME**

50494 wctomb — convert a wide-character code to a character

50495 **SYNOPSIS**

50496 #include <stdlib.h>

50497 int wctomb(char *s, wchar_t wchar);

50498 **DESCRIPTION**

50499 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 50500 conflict between the requirements described here and the ISO C standard is unintentional. This
 50501 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50502 The *wctomb()* function shall determine the number of bytes needed to represent the character
 50503 corresponding to the wide-character code whose value is *wchar* (including any change in the
 50504 shift state). It shall store the character representation (possibly multiple bytes and any special
 50505 bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most
 50506 {MB_CUR_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any
 50507 shift sequence needed to restore the initial shift state, and *wctomb()* shall be left in the initial shift
 50508 state.

50509 CX The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 50510 state-dependent encoding, this function shall be placed into its initial state by a call for which its
 50511 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 50512 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 50513 null pointer shall cause this function to return a non-zero value if encodings have state
 50514 dependency, and 0 otherwise. Changing the *LC_CTYPE* category causes the shift state of this
 50515 function to be unspecified.

50516 The *wctomb()* function need not be reentrant. A function that is not required to be reentrant is
 50517 not required to be thread-safe.

50518 The implementation shall behave as if no function defined in this volume of
 50519 IEEE Std 1003.1-2001 calls *wctomb()*.

50520 **RETURN VALUE**

50521 If *s* is a null pointer, *wctomb()* shall return a non-zero or 0 value, if character encodings,
 50522 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb()*
 50523 shall return -1 if the value of *wchar* does not correspond to a valid character, or return the
 50524 number of bytes that constitute the character corresponding to the value of *wchar*.

50525 In no case shall the value returned be greater than the value of the {MB_CUR_MAX} macro.

50526 **ERRORS**

50527 No errors are defined.

50528 **EXAMPLES**

50529 None.

50530 **APPLICATION USAGE**

50531 None.

50532 **RATIONALE**

50533 None.

50534 **FUTURE DIRECTIONS**

50535 None.

50536 **SEE ALSO**

50537 *mblen()*, *mbtowc()*, *mbstowcs()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-2001,
50538 **<stdlib.h>**

50539 **CHANGE HISTORY**

50540 First released in Issue 4. Derived from the ANSI C standard.

50541 **Issue 6**

50542 Extensions beyond the ISO C standard are marked.

50543 In the DESCRIPTION, a note about reentrancy and thread-safety is added.

50544 **NAME**

50545 wctrans — define character mapping

50546 **SYNOPSIS**

50547 #include <wctype.h>

50548 wctrans_t wctrans(const char *charclass);

50549 **DESCRIPTION**

50550 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 50551 conflict between the requirements described here and the ISO C standard is unintentional. This
 50552 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50553 The *wctrans()* function is defined for valid character mapping names identified in the current
 50554 locale. The *charclass* is a string identifying a generic character mapping name for which codeset-
 50555 specific information is required. The following character mapping names are defined in all
 50556 locales: **tolower** and **toupper**.

50557 The function shall return a value of type **wctrans_t**, which can be used as the second argument
 50558 to subsequent calls of *towctrans()*. The *wctrans()* function shall determine values of **wctrans_t**
 50559 according to the rules of the coded character set defined by character mapping information in
 50560 the program's locale (category *LC_CTYPE*). The values returned by *wctrans()* shall be valid until
 50561 a call to *setlocale()* that modifies the category *LC_CTYPE*.

50562 **RETURN VALUE**

50563 CX The *wctrans()* function shall return 0 and may set *errno* to indicate the error if the given
 50564 character mapping name is not valid for the current locale (category *LC_CTYPE*); otherwise, it
 50565 shall return a non-zero object of type **wctrans_t** that can be used in calls to *towctrans()*.

50566 **ERRORS**50567 The *wctrans()* function may fail if:

50568 CX [EINVAL] The character mapping name pointed to by *charclass* is not valid in the current
 50569 locale.

50570 **EXAMPLES**

50571 None.

50572 **APPLICATION USAGE**

50573 None.

50574 **RATIONALE**

50575 None.

50576 **FUTURE DIRECTIONS**

50577 None.

50578 **SEE ALSO**50579 *towctrans()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wctype.h**>50580 **CHANGE HISTORY**

50581 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

50582 **NAME**

50583 wctype — define character class

50584 **SYNOPSIS**

50585 #include <wctype.h>

50586 wctype_t wctype(const char *property);

50587 **DESCRIPTION**

50588 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 50589 conflict between the requirements described here and the ISO C standard is unintentional. This
 50590 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50591 The *wctype()* function is defined for valid character class names as defined in the current locale.
 50592 The *property* argument is a string identifying a generic character class for which codeset-specific
 50593 type information is required. The following character class names shall be defined in all locales:

50594	alnum	digit	punct
50595	alpha	graph	space
50596	blank	lower	upper
50597	cntrl	print	xdigit

50598 Additional character class names defined in the locale definition file (category *LC_CTYPE*) can
 50599 also be specified.

50600 The function shall return a value of type **wctype_t**, which can be used as the second argument to
 50601 subsequent calls of *iswctype()*. The *wctype()* function shall determine values of **wctype_t**
 50602 according to the rules of the coded character set defined by character type information in the
 50603 program's locale (category *LC_CTYPE*). The values returned by *wctype()* shall be valid until a
 50604 call to *setlocale()* that modifies the category *LC_CTYPE*.

50605 **RETURN VALUE**

50606 The *wctype()* function shall return 0 if the given character class name is not valid for the current
 50607 locale (category *LC_CTYPE*); otherwise, it shall return an object of type **wctype_t** that can be
 50608 used in calls to *iswctype()*.

50609 **ERRORS**

50610 No errors are defined.

50611 **EXAMPLES**

50612 None.

50613 **APPLICATION USAGE**

50614 None.

50615 **RATIONALE**

50616 None.

50617 **FUTURE DIRECTIONS**

50618 None.

50619 **SEE ALSO**50620 *iswctype()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wctype.h**>50621 **CHANGE HISTORY**

50622 First released in Issue 4.

50623 **Issue 5**

50624 The following change has been made in this issue for alignment with
50625 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 50626 • The SYNOPSIS has been changed to indicate that this function and associated data types are
50627 now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

50628 **NAME**

50629 wctype — number of column positions of a wide-character code

50630 **SYNOPSIS**

50631 xSI #include <wctype.h>

50632 int wctype(wctype_t wc);

50633

50634 **DESCRIPTION**

50635 The wctype() function shall determine the number of column positions required for the wide
50636 character wc. The application shall ensure that the value of wc is a character representable as a
50637 **wctype_t**, and is a wide-character code corresponding to a valid character in the current locale.

50638 **RETURN VALUE**

50639 The wctype() function shall either return 0 (if wc is a null wide-character code), or return the
50640 number of column positions to be occupied by the wide-character code wc, or return -1 (if wc
50641 does not correspond to a printable wide-character code).

50642 **ERRORS**

50643 No errors are defined.

50644 **EXAMPLES**

50645 None.

50646 **APPLICATION USAGE**

50647 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the
50648 return value for a non-printable wide character is not specified.

50649 **RATIONALE**

50650 None.

50651 **FUTURE DIRECTIONS**

50652 None.

50653 **SEE ALSO**

50654 wctype(), the Base Definitions volume of IEEE Std 1003.1-2001, <wctype.h>

50655 **CHANGE HISTORY**

50656 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
50657 draft.

50658 **Issue 6**

50659 The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

50660 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50661 **NAME**

50662 wmemchr — find a wide character in memory

50663 **SYNOPSIS**

50664 #include <wchar.h>

50665 wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);

50666 **DESCRIPTION**

50667 cx The functionality described on this reference page is aligned with the ISO C standard. Any
50668 conflict between the requirements described here and the ISO C standard is unintentional. This
50669 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50670 The *wmemchr()* function shall locate the first occurrence of *wc* in the initial *n* wide characters of
50671 the object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values
50672 shall be treated identically. The null wide character and **wchar_t** values not corresponding to
50673 valid characters shall not be treated specially.

50674 If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if
50675 no valid occurrence of *wc* is found.

50676 **RETURN VALUE**

50677 The *wmemchr()* function shall return a pointer to the located wide character, or a null pointer if
50678 the wide character does not occur in the object.

50679 **ERRORS**

50680 No errors are defined.

50681 **EXAMPLES**

50682 None.

50683 **APPLICATION USAGE**

50684 None.

50685 **RATIONALE**

50686 None.

50687 **FUTURE DIRECTIONS**

50688 None.

50689 **SEE ALSO**

50690 *wmemcmp()*, *wmemcpy()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of
50691 IEEE Std 1003.1-2001, <**wchar.h**>

50692 **CHANGE HISTORY**

50693 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
50694 (E).

50695 **Issue 6**

50696 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50697 **NAME**

50698 wmemcmp — compare wide characters in memory

50699 **SYNOPSIS**

50700 #include <wchar.h>

50701 int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);

50702 **DESCRIPTION**

50703 cx The functionality described on this reference page is aligned with the ISO C standard. Any
50704 conflict between the requirements described here and the ISO C standard is unintentional. This
50705 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50706 The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by
50707 *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be
50708 affected by locale and all **wchar_t** values shall be treated identically. The null wide character and
50709 **wchar_t** values not corresponding to valid characters shall not be treated specially.

50710 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
50711 shall behave as if the two objects compare equal.

50712 **RETURN VALUE**

50713 The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero,
50714 respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object
50715 pointed to by *ws2*.

50716 **ERRORS**

50717 No errors are defined.

50718 **EXAMPLES**

50719 None.

50720 **APPLICATION USAGE**

50721 None.

50722 **RATIONALE**

50723 None.

50724 **FUTURE DIRECTIONS**

50725 None.

50726 **SEE ALSO**

50727 *wmemchr()*, *wmemcpy()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of
50728 IEEE Std 1003.1-2001, <**wchar.h**>

50729 **CHANGE HISTORY**

50730 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
50731 (E).

50732 **Issue 6**

50733 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50734 **NAME**

50735 wmemcpy — copy wide characters in memory

50736 **SYNOPSIS**

50737 #include <wchar.h>

```
50738 wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,  
50739                 size_t n);
```

50740 **DESCRIPTION**

50741 cx The functionality described on this reference page is aligned with the ISO C standard. Any
50742 conflict between the requirements described here and the ISO C standard is unintentional. This
50743 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50744 The `wmemcpy()` function shall copy *n* wide characters from the object pointed to by `ws2` to the
50745 object pointed to by `ws1`. This function shall not be affected by locale and all **wchar_t** values
50746 shall be treated identically. The null wide character and **wchar_t** values not corresponding to
50747 valid characters shall not be treated specially.

50748 If *n* is zero, the application shall ensure that `ws1` and `ws2` are valid pointers, and the function
50749 shall copy zero wide characters.

50750 **RETURN VALUE**50751 The `wmemcpy()` function shall return the value of `ws1`.50752 **ERRORS**

50753 No errors are defined.

50754 **EXAMPLES**

50755 None.

50756 **APPLICATION USAGE**

50757 None.

50758 **RATIONALE**

50759 None.

50760 **FUTURE DIRECTIONS**

50761 None.

50762 **SEE ALSO**

50763 `wmemchr()`, `wmemcmp()`, `wmemmove()`, `wmemset()`, the Base Definitions volume of
50764 IEEE Std 1003.1-2001, <**wchar.h**>

50765 **CHANGE HISTORY**

50766 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
50767 (E).

50768 **Issue 6**

50769 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50770 The `wmemcpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

50771 NAME

50772 wmemmove — copy wide characters in memory with overlapping areas

50773 SYNOPSIS

50774 #include <wchar.h>

50775 wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);

50776 DESCRIPTION

50777 cx The functionality described on this reference page is aligned with the ISO C standard. Any
50778 conflict between the requirements described here and the ISO C standard is unintentional. This
50779 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50780 The *wmemmove()* function shall copy *n* wide characters from the object pointed to by *ws2* to the
50781 object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object
50782 pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not
50783 overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary
50784 array are copied into the object pointed to by *ws1*.

50785 This function shall not be affected by locale and all **wchar_t** values shall be treated identically.
50786 The null wide character and **wchar_t** values not corresponding to valid characters shall not be
50787 treated specially.

50788 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
50789 shall copy zero wide characters.

50790 RETURN VALUE

50791 The *wmemmove()* function shall return the value of *ws1*.

50792 ERRORS

50793 No errors are defined

50794 EXAMPLES

50795 None.

50796 APPLICATION USAGE

50797 None.

50798 RATIONALE

50799 None.

50800 FUTURE DIRECTIONS

50801 None.

50802 SEE ALSO

50803 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemset()*, the Base Definitions volume of
50804 IEEE Std 1003.1-2001, <**wchar.h**>

50805 CHANGE HISTORY

50806 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
50807 (E).

50808 Issue 6

50809 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50810 **NAME**

50811 wmemset — set wide characters in memory

50812 **SYNOPSIS**

50813 #include <wchar.h>

50814 wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);

50815 **DESCRIPTION**

50816 cx The functionality described on this reference page is aligned with the ISO C standard. Any
50817 conflict between the requirements described here and the ISO C standard is unintentional. This
50818 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50819 The *wmemset()* function shall copy the value of *wc* into each of the first *n* wide characters of the
50820 object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values shall
50821 be treated identically. The null wide character and **wchar_t** values not corresponding to valid
50822 characters shall not be treated specially.

50823 If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function shall copy
50824 zero wide characters.

50825 **RETURN VALUE**50826 The *wmemset()* functions shall return the value of *ws*.50827 **ERRORS**

50828 No errors are defined.

50829 **EXAMPLES**

50830 None.

50831 **APPLICATION USAGE**

50832 None.

50833 **RATIONALE**

50834 None.

50835 **FUTURE DIRECTIONS**

50836 None.

50837 **SEE ALSO**

50838 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemmove()*, the Base Definitions volume of
50839 IEEE Std 1003.1-2001, <**wchar.h**>

50840 **CHANGE HISTORY**

50841 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
50842 (E).

50843 **Issue 6**

50844 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50845 **NAME**

50846 wordexp, wordfree — perform word expansions

50847 **SYNOPSIS**

50848 #include <wordexp.h>

50849 int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
50850 int flags);

50851 void wordfree(wordexp_t *pwordexp);

50852 **DESCRIPTION**

50853 The *wordexp()* function shall perform word expansions as described in the Shell and Utilities
 50854 volume of IEEE Std 1003.1-2001, Section 2.6, Word Expansions, subject to quoting as in the Shell
 50855 and Utilities volume of IEEE Std 1003.1-2001, Section 2.2, Quoting, and place the list of expanded
 50856 words into the structure pointed to by *pwordexp*.

50857 The *words* argument is a pointer to a string containing one or more words to be expanded. The
 50858 expansions shall be the same as would be performed by the command line interpreter if *words*
 50859 were the part of a command line representing the arguments to a utility. Therefore, the
 50860 application shall ensure that *words* does not contain an unquoted <newline> or any of the
 50861 unquoted shell special characters ' | ', ' & ', ' ; ', ' < ', ' > ' except in the context of command
 50862 substitution as specified in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.3,
 50863 Command Substitution. It also shall not contain unquoted parentheses or braces, except in the
 50864 context of command or variable substitution. The application shall ensure that every member of
 50865 *words* which it expects to have expanded by *wordexp()* does not contain an unquoted initial
 50866 comment character. The application shall also ensure that any words which it intends to be
 50867 ignored (because they begin or continue a comment) are deleted from *words*. If the argument
 50868 *words* contains an unquoted comment character (number sign) that is the beginning of a token,
 50869 *wordexp()* shall either treat the comment character as a regular character, or interpret it as a
 50870 comment indicator and ignore the remainder of *words*.

50871 The structure type **wordexp_t** is defined in the <**wordexp.h**> header and includes at least the
 50872 following members:

50873

50874

50875

50876

50877

Member Type	Member Name	Description
size_t	<i>we_wordc</i>	Count of words matched by <i>words</i> .
char **	<i>we_wordv</i>	Pointer to list of expanded words.
size_t	<i>we_offs</i>	Slots to reserve at the beginning of <i>pwordexp->we_wordv</i> .

50878 The *wordexp()* function shall store the number of generated words into *pwordexp->we_wordc* and
 50879 a pointer to a list of pointers to words in *pwordexp->we_wordv*. Each individual field created
 50880 during field splitting (see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.5,
 50881 Field Splitting) or pathname expansion (see the Shell and Utilities volume of
 50882 IEEE Std 1003.1-2001, Section 2.6.6, Pathname Expansion) shall be a separate word in the
 50883 *pwordexp->we_wordv* list. The words shall be in order as described in the Shell and Utilities
 50884 volume of IEEE Std 1003.1-2001, Section 2.6, Word Expansions. The first pointer after the last
 50885 word pointer shall be a null pointer. The expansion of special parameters described in the Shell
 50886 and Utilities volume of IEEE Std 1003.1-2001, Section 2.5.2, Special Parameters is unspecified.

50887 It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()*
 50888 function shall allocate other space as needed, including memory pointed to by
 50889 *pwordexp->we_wordv*. The *wordfree()* function frees any memory associated with *pwordexp* from a
 50890 previous call to *wordexp()*.

50891 The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the
 50892 bitwise-inclusive OR of zero or more of the following constants, which are defined in
 50893 **<wordexp.h>**:

50894	WRDE_APPEND	Append words generated to the ones from a previous call to <i>wordexp()</i> .
50895	WRDE_DOOFFS	Make use of <i>pwordexp->we_offs</i> . If this flag is set, <i>pwordexp->we_offs</i> is used
50896		to specify how many null pointers to add to the beginning of
50897		<i>pwordexp->we_wordv</i> . In other words, <i>pwordexp->we_wordv</i> shall point to
50898		<i>pwordexp->we_offs</i> null pointers, followed by <i>pwordexp->we_wordc</i> word
50899		pointers, followed by a null pointer.
50900	WRDE_NOCMD	If the implementation supports the utilities defined in the Shell and
50901		Utilities volume of IEEE Std 1003.1-2001, fail if command substitution, as
50902		specified in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
50903		2.6.3, Command Substitution, is requested.
50904	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to
50905		<i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result shall be the
50906		same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i>
50907		without WRDE_REUSE.
50908	WRDE_SHOWERR	Do not redirect <i>stderr</i> to /dev/null .
50909	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.

50910 The WRDE_APPEND flag can be used to append a new set of words to those generated by a
 50911 previous call to *wordexp()*. The following rules apply to applications when two or more calls to
 50912 *wordexp()* are made with the same value of *pwordexp* and without intervening calls to *wordfree()*:

- 50913 1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.
- 50914 2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.
- 50915 3. After the second and each subsequent call, *pwordexp->we_wordv* shall point to a list
 50916 containing the following:
 - 50917 a. Zero or more null pointers, as specified by WRDE_DOOFFS and *pwordexp->we_offs*
 - 50918 b. Pointers to the words that were in the *pwordexp->we_wordv* list before the call, in the
 50919 same order as before
 - 50920 c. Pointers to the new words generated by the latest call, in the specified order
- 50921 4. The count returned in *pwordexp->we_wordc* shall be the total number of words from all of
 50922 the calls.
- 50923 5. The application can change any of the fields after a call to *wordexp()*, but if it does it shall
 50924 reset them to the original value before a subsequent call, using the same *pwordexp* value, to
 50925 *wordfree()* or *wordexp()* with the WRDE_APPEND or WRDE_REUSE flag.

50926 If the implementation supports the utilities defined in the Shell and Utilities volume of
 50927 IEEE Std 1003.1-2001, and *words* contains an unquoted character—<newline>, ' | ', ' & ', ' ; ',
 50928 ' < ', ' > ', ' (', ') ', ' { ', ' } '—in an inappropriate context, *wordexp()* shall fail, and the number
 50929 of expanded words shall be 0.

50930 Unless WRDE_SHOWERR is set in *flags*, *wordexp()* shall redirect *stderr* to **/dev/null** for any
 50931 utilities executed as a result of command substitution while expanding *words*. If
 50932 WRDE_SHOWERR is set, *wordexp()* may write messages to *stderr* if syntax errors are detected
 50933 while expanding *words*.

50934 The application shall ensure that if WRDE_DOOFFS is set, then *pwordexp->we_offs* has the same
 50935 value for each *wordexp()* call and *wordfree()* call using a given *pwordexp*.

50936 The following constants are defined as error return values:

50937 WRDE_BADCHAR One of the unquoted characters—<newline>, ' | ', ' & ', ' ; ', ' < ', ' > ',
 50938 ' (', ') ', ' { ', ' } '—appears in *words* in an inappropriate context.

50939 WRDE_BADVAL Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.

50940 WRDE_CMDSUB Command substitution requested when WRDE_NOCMD was set in *flags*.

50941 WRDE_NOSPACE Attempt to allocate memory failed.

50942 WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated
 50943 string.

50944 RETURN VALUE

50945 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described
 50946 in <**wordexp.h**>, shall be returned to indicate an error. If *wordexp()* returns the value
 50947 WRDE_NOSPACE, then *pwordexp->we_wordc* and *pwordexp->we_wordv* shall be updated to
 50948 reflect any words that were successfully expanded. In other cases, they shall not be modified.

50949 The *wordfree()* function shall not return a value.

50950 ERRORS

50951 No errors are defined.

50952 EXAMPLES

50953 None.

50954 APPLICATION USAGE

50955 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's
 50956 expansions on a word or words obtained from a user. For example, if the application prompts
 50957 for a filename (or list of filenames) and then uses *wordexp()* to process the input, the user could
 50958 respond with anything that would be valid as input to the shell.

50959 The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to
 50960 prevent a user from executing shell commands. Disallowing unquoted shell special characters
 50961 also prevents unwanted side effects, such as executing a command or writing a file.

50962 RATIONALE

50963 This function was included as an alternative to *glob()*. There had been continuing controversy
 50964 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*
 50965 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()*
 50966 (which is faster, but which only performs pathname expansion, without tilde or parameter
 50967 expansion) this will satisfy the majority of applications.

50968 While *wordexp()* could be implemented entirely as a library routine, it is expected that most
 50969 implementations run a shell in a subprocess to do the expansion.

50970 Two different approaches have been proposed for how the required information might be
 50971 presented to the shell and the results returned. They are presented here as examples.

50972 One proposal is to extend the *echo* utility by adding a **-q** option. This option would cause *echo* to
 50973 add a backslash before each backslash and <blank> that occurs within an argument. The
 50974 *wordexp()* function could then invoke the shell as follows:

```
50975 (void) strcpy(buffer, "echo -q");
50976 (void) strcat(buffer, words);
50977 if ((flags & WRDE_SHOWERR) == 0)
```

```

50978         (void) strcat(buffer, "2>/dev/null");
50979         f = popen(buffer, "r");

```

50980 The *wordexp()* function would read the resulting output, remove unquoted backslashes, and
 50981 break into words at unquoted <blank>s. If the WRDE_NOCMD flag was set, *wordexp()* would
 50982 have to scan *words* before starting the subshell to make sure that there would be no command
 50983 substitution. In any case, it would have to scan *words* for unquoted special characters.

50984 Another proposal is to add the following options to *sh*:

50985 **-w wordlist**

50986 This option provides a wordlist expansion service to applications. The words in *wordlist*
 50987 shall be expanded and the following written to standard output:

- 50988 1. The count of the number of words after expansion, in decimal, followed by a null byte
- 50989 2. The number of bytes needed to represent the expanded words (not including null
 50990 separators), in decimal, followed by a null byte
- 50991 3. The expanded words, each terminated by a null byte

50992 If an error is encountered during word expansion, *sh* exits with a non-zero status after
 50993 writing the former to report any words successfully expanded

50994 **-P** Run in “protected” mode. If specified with the **-w** option, no command substitution shall
 50995 be performed.

50996 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess
 50997 using *fork()* and executing *sh* using the line:

```

50998 execl(<shell path>, "sh", "-P", "-w", words, (char *)0);

```

50999 after directing standard error to **/dev/null**.

51000 It seemed objectionable for a library routine to write messages to standard error, unless
 51001 explicitly requested, so *wordexp()* is required to redirect standard error to **/dev/null** to ensure
 51002 that no messages are generated, even for commands executed for command substitution. The
 51003 WRDE_SHOWERR flag can be specified to request that error messages be written.

51004 The WRDE_REUSE flag allows the implementation to avoid the expense of freeing and
 51005 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when
 51006 WRDE_REUSE is set.

51007 FUTURE DIRECTIONS

51008 None.

51009 SEE ALSO

51010 *fnmatch()*, *glob()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<wordexp.h>**, the Shell
 51011 and Utilities volume of IEEE Std 1003.1-2001, Chapter 2, Shell Command Language

51012 CHANGE HISTORY

51013 First released in Issue 4. Derived from the ISO POSIX-2 standard.

51014 Issue 5

51015 Moved from POSIX2 C-language Binding to BASE.

51016 Issue 6

51017 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

51018 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the
 51019 ISO/IEC 9899:1999 standard.

51020 **NAME**51021 **wprintf** — print formatted wide-character output51022 **SYNOPSIS**

51023 #include <stdio.h>

51024 #include <wchar.h>

51025 int wprintf(const wchar_t *restrict *format*, ...);51026 **DESCRIPTION**51027 Refer to *fwprintf()*.

51028 **NAME**

51029 pwrite, write — write on a file

51030 **SYNOPSIS**

51031 #include <unistd.h>

```
51032 xsi ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
51033         off_t offset);
51034 ssize_t write(int fildes, const void *buf, size_t nbyte);
```

51035 **DESCRIPTION**

51036 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the
 51037 file associated with the open file descriptor, *fildes*.

51038 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the
 51039 *write()* function may detect and return errors as described below. In the absence of errors, or if
 51040 error detection is not performed, the *write()* function shall return zero and have no other results.
 51041 If *nbyte* is zero and the file is not a regular file, the results are unspecified.

51042 On a regular file or other file capable of seeking, the actual writing of data shall proceed from the
 51043 position in the file indicated by the file offset associated with *fildes*. Before successful return
 51044 from *write()*, the file offset shall be incremented by the number of bytes actually written. On a
 51045 regular file, if this incremented file offset is greater than the length of the file, the length of the
 51046 file shall be set to this file offset.

51047 On a file not capable of seeking, writing shall always take place starting at the current position.
 51048 The value of a file offset associated with such a device is undefined.

51049 If the O_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file
 51050 prior to each write and no intervening file modification operation shall occur between changing
 51051 the file offset and the write operation.

51052 xsi If a *write()* requests that more bytes be written than there is room for (for example, the process'
 51053 file size limit or the physical end of a medium), only as many bytes as there is room for shall be
 51054 written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A
 51055 write of 512 bytes will return 20. The next write of a non-zero number of bytes would give a
 51056 failure return (except as noted below).

51057 xsi If the request would cause the file size to exceed the soft file size limit for the process and there
 51058 is no room for any bytes to be written, the request shall fail and the implementation shall
 51059 generate the SIGXFSZ signal for the thread.

51060 If *write()* is interrupted by a signal before it writes any data, it shall return -1 with *errno* set to
 51061 [EINTR].

51062 If *write()* is interrupted by a signal after it successfully writes some data, it shall return the
 51063 number of bytes written.

51064 If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

51065 After a *write()* to a regular file has successfully returned:

- 51066 • Any successful *read()* from each byte position in the file that was modified by that write shall
 51067 return the data specified by the *write()* for that position until such byte positions are again
 51068 modified.
- 51069 • Any subsequent successful *write()* to the same byte position in the file shall overwrite that
 51070 file data.

51071	Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the
51072	following exceptions:
51073	• There is no file offset associated with a pipe, hence each write request shall append to the
51074	end of the pipe.
51075	• Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other
51076	processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may have
51077	data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the
51078	O_NONBLOCK flag of the file status flags is set.
51079	• If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on
51080	normal completion it shall return <i>nbyte</i> .
51081	• If the O_NONBLOCK flag is set, <i>write()</i> requests shall be handled differently, in the
51082	following ways:
51083	— The <i>write()</i> function shall not block the thread.
51084	— A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there is
51085	sufficient space available in the pipe, <i>write()</i> shall transfer all the data and return the
51086	number of bytes requested. Otherwise, <i>write()</i> shall transfer no data and return <i>-1</i> with
51087	<i>errno</i> set to [EAGAIN].
51088	— A write request for more than {PIPE_BUF} bytes shall cause one of the following:
51089	— When at least one byte can be written, transfer what it can and return the number of
51090	bytes written. When all data previously written to the pipe is read, it shall transfer at
51091	least {PIPE_BUF} bytes.
51092	— When no data can be written, transfer no data, and return <i>-1</i> with <i>errno</i> set to
51093	[EAGAIN].
51094	When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-
51095	blocking writes and cannot accept the data immediately:
51096	• If the O_NONBLOCK flag is clear, <i>write()</i> shall block the calling thread until the data can be
51097	accepted.
51098	• If the O_NONBLOCK flag is set, <i>write()</i> shall not block the thread. If some data can be
51099	written without blocking the thread, <i>write()</i> shall write what it can and return the number of
51100	bytes written. Otherwise, it shall return <i>-1</i> and set <i>errno</i> to [EAGAIN].
51101	Upon successful completion, where <i>nbyte</i> is greater than 0, <i>write()</i> shall mark for update the
51102	<i>st_ctime</i> and <i>st_mtime</i> fields of the file, and if the file is a regular file, the S_ISUID and S_ISGID
51103	bits of the file mode may be cleared.
51104	For regular files, no data transfer shall occur past the offset maximum established in the open
51105	file description associated with <i>fildes</i> .
51106	If <i>fildes</i> refers to a socket, <i>write()</i> shall be equivalent to <i>send()</i> with no flags set.
51107 SIO	If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as
51108	defined by synchronized I/O data integrity completion.
51109	If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as
51110	defined by synchronized I/O file integrity completion.
51111 SHM	If <i>fildes</i> refers to a shared memory object, the result of the <i>write()</i> function is unspecified.
51112 TYM	If <i>fildes</i> refers to a typed memory object, the result of the <i>write()</i> function is unspecified.

51113 XSR If *fildev* refers to a STREAM, the operation of *write()* shall be determined by the values of the
 51114 minimum and maximum *nbyte* range (packet size) accepted by the STREAM. These values are
 51115 determined by the topmost STREAM module. If *nbyte* falls within the packet size range, *nbyte*
 51116 bytes shall be written. If *nbyte* does not fall within the range and the minimum packet size value
 51117 is 0, *write()* shall break the buffer into maximum packet size segments prior to sending the data
 51118 downstream (the last segment may contain less than the maximum packet size). If *nbyte* does not
 51119 fall within the range and the minimum value is non-zero, *write()* shall fail with *errno* set to
 51120 [ERANGE]. Writing a zero-length buffer (*nbyte* is 0) to a STREAMS device sends 0 bytes with 0
 51121 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no
 51122 message and 0 is returned. The process may issue *I_SWROPT ioctl()* to enable zero-length
 51123 messages to be sent across the pipe or FIFO.

51124 When writing to a STREAM, data messages are created with a priority band of 0. When writing
 51125 to a STREAM that is not a pipe or FIFO:

51126 • If O_NONBLOCK is clear, and the STREAM cannot accept data (the STREAM write queue is
 51127 full due to internal flow control conditions), *write()* shall block until data can be accepted.

51128 • If O_NONBLOCK is set and the STREAM cannot accept data, *write()* shall return -1 and set
 51129 *errno* to [EAGAIN].

51130 • If O_NONBLOCK is set and part of the buffer has been written while a condition in which
 51131 the STREAM cannot accept additional data occurs, *write()* shall terminate and return the
 51132 number of bytes written.

51133 In addition, *write()* shall fail if the STREAM head has processed an asynchronous error before
 51134 the call. In this case, the value of *errno* does not reflect the result of *write()*, but reflects the prior
 51135 error.

51136 XSI The *pwrite()* function shall be equivalent to *write()*, except that it writes into a given position
 51137 without changing the file pointer. The first three arguments to *pwrite()* are the same as *write()*
 51138 with the addition of a fourth argument offset for the desired position inside the file.

51139 RETURN VALUE

51140 XSI Upon successful completion, *write()* and *pwrite()* shall return the number of bytes actually
 51141 written to the file associated with *fildev*. This number shall never be greater than *nbyte*.
 51142 Otherwise, -1 shall be returned and *errno* set to indicate the error.

51143 ERRORS

51144 XSI The *write()* and *pwrite()* functions shall fail if:

51145 [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the thread would be
 51146 delayed in the *write()* operation.

51147 [EBADF] The *fildev* argument is not a valid file descriptor open for writing.

51148 [EFBIG] An attempt was made to write a file that exceeds the implementation-defined
 51149 XSI maximum file size or the process' file size limit, and there was no room for
 51150 any bytes to be written.

51151 [EFBIG] The file is a regular file, *nbyte* is greater than 0, and the starting position is
 51152 greater than or equal to the offset maximum established in the open file
 51153 description associated with *fildev*.

51154 [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 51155 was transferred.

51156 [EIO] The process is a member of a background process group attempting to write
 51157 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor

51158		blocking SIGTTOU, and the process group of the process is orphaned. This
51159		error may also be returned under implementation-defined conditions.
51160	[ENOSPC]	There was no free space remaining on the device containing the file.
51161	[EPIPE]	An attempt is made to write to a pipe or FIFO that is not open for reading by
51162		any process, or that only has one end open. A SIGPIPE signal shall also be sent
51163		to the thread.
51164 XSR	[ERANGE]	The transfer request size was outside the range supported by the STREAMS
51165		file associated with <i>fildev</i> .
51166		The <i>write()</i> function shall fail if:
51167	[EAGAIN] or [EWOULDBLOCK]	
51168		The file descriptor is for a socket, is marked O_NONBLOCK, and write would
51169		block.
51170	[ECONNRESET]	A write was attempted on a socket that is not connected.
51171	[EPIPE]	A write was attempted on a socket that is shut down for writing, or is no
51172		longer connected. In the latter case, if the socket is of type SOCK_STREAM,
51173		the SIGPIPE signal is generated to the calling process.
51174 XSI		The <i>write()</i> and <i>pwrite()</i> functions may fail if:
51175 XSR	[EINVAL]	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or
51176		indirectly) downstream from a multiplexer.
51177	[EIO]	A physical I/O error has occurred.
51178	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
51179	[ENXIO]	A request was made of a nonexistent device, or the request was outside the
51180		capabilities of the device.
51181 XSR	[ENXIO]	A hangup occurred on the STREAM being written to.
51182 XSR		A write to a STREAMS file may fail if an error message has been received at the STREAM head.
51183		In this case, <i>errno</i> is set to the value included in the error message.
51184		The <i>write()</i> function may fail if:
51185	[EACCES]	A write was attempted on a socket and the calling process does not have
51186		appropriate privileges.
51187	[ENETDOWN]	A write was attempted on a socket and the local network interface used to
51188		reach the destination is down.
51189	[ENETUNREACH]	
51190		A write was attempted on a socket and no route to the network is present.
51191 XSI		The <i>pwrite()</i> function shall fail and the file pointer remain unchanged if:
51192 XSI	[EINVAL]	The <i>offset</i> argument is invalid. The value is negative.
51193 XSI	[ESPIPE]	<i>fildev</i> is associated with a pipe or FIFO.

51194 **EXAMPLES**51195 **Writing from a Buffer**

51196 The following example writes data from the buffer pointed to by *buf* to the file associated with
 51197 the file descriptor *fd*.

```

51198 #include <sys/types.h>
51199 #include <string.h>
51200 ...
51201 char buf[20];
51202 size_t nbytes;
51203 ssize_t bytes_written;
51204 int fd;
51205 ...
51206 strcpy(buf, "This is a test\n");
51207 nbytes = strlen(buf);

51208 bytes_written = write(fd, buf, nbytes);
51209 ...

```

51210 **APPLICATION USAGE**

51211 None.

51212 **RATIONALE**

51213 See also the RATIONALE section in *read()*.

51214 An attempt to write to a pipe or FIFO has several major characteristics:

- 51215 • *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not
 51216 interleaved with data from any other process. This is useful when there are multiple writers
 51217 sending data to a single reader. Applications need to know how large a write request can be
 51218 expected to be performed atomically. This maximum is called {PIPE_BUF}. This volume of
 51219 IEEE Std 1003.1-2001 does not say whether write requests for more than {PIPE_BUF} bytes
 51220 are atomic, but requires that writes of {PIPE_BUF} or fewer bytes shall be atomic.
- 51221 • *Blocking/immediate*: Blocking is only possible with O_NONBLOCK clear. If there is enough
 51222 space for all the data requested to be written immediately, the implementation should do so.
 51223 Otherwise, the process may block; that is, pause until enough space is available for writing.
 51224 The effective size of a pipe or FIFO (the maximum amount that can be written in one
 51225 operation without blocking) may vary dynamically, depending on the implementation, so it
 51226 is not possible to specify a fixed value for it.

- 51227 • *Complete/partial/deferred*: A write request:

```

51228 int fildes;
51229 size_t nbyte;
51230 ssize_t ret;
51231 char *buf;

```

```

51232 ret = write(fildes, buf, nbyte);

```

51233 may return:

51234 Complete *ret*=*nbyte*

51235 Partial *ret*<*nbyte*

51236 This shall never happen if *nbyte*≤{PIPE_BUF}. If it does happen (with
 51237 *nbyte*>{PIPE_BUF}), this volume of IEEE Std 1003.1-2001 does not guarantee

51238 atomicity, even if $ret \leq \{PIPE\_BUF\}$, because atomicity is guaranteed according
 51239 to the amount *requested*, not the amount *written*.

51240 Deferred: $ret = -1$, $errno = [EAGAIN]$

51241 This error indicates that a later request may succeed. It does not indicate that it
 51242 *shall* succeed, even if $nbyte \leq \{PIPE\_BUF\}$, because if no process reads from the
 51243 pipe or FIFO, the write never succeeds. An application could usefully count the
 51244 number of times $[EAGAIN]$ is caused by a particular value of
 51245 $nbyte > \{PIPE\_BUF\}$ and perhaps do later writes with a smaller value, on the
 51246 assumption that the effective size of the pipe may have decreased.

51247 Partial and deferred writes are only possible with $O\_NONBLOCK$ set.

51248 The relations of these properties are shown in the following tables:

51249

51250

51251

51252

51253

51254

Write to a Pipe or FIFO with $O\_NONBLOCK$ <i>clear</i>			
Immediately Writable:	None	Some	$nbyte$
$nbyte \leq \{PIPE\_BUF\}$	Atomic blocking $nbyte$	Atomic blocking $nbyte$	Atomic immediate $nbyte$
$nbyte > \{PIPE\_BUF\}$	Blocking $nbyte$	Blocking $nbyte$	Blocking $nbyte$

51255 If the $O\_NONBLOCK$ flag is clear, a write request shall block if the amount writable
 51256 immediately is less than that requested. If the flag is set (by $fctl()$), a write request shall never
 51257 block.

51258

51259

51260

51261

51262

51263

Write to a Pipe or FIFO with $O\_NONBLOCK$ <i>set</i>			
Immediately Writable:	None	Some	$nbyte$
$nbyte \leq \{PIPE\_BUF\}$	$-1, [EAGAIN]$	$-1, [EAGAIN]$	Atomic $nbyte$
$nbyte > \{PIPE\_BUF\}$	$-1, [EAGAIN]$	$<nbyte$ or $-1,$ [EAGAIN]	$\leq nbyte$ or $-1,$ [EAGAIN]

51264 There is no exception regarding partial writes when $O\_NONBLOCK$ is set. With the exception
 51265 of writing to an empty pipe, this volume of IEEE Std 1003.1-2001 does not specify exactly when a
 51266 partial write is performed since that would require specifying internal details of the
 51267 implementation. Every application should be prepared to handle partial writes when
 51268 $O\_NONBLOCK$ is set and the requested amount is greater than $\{PIPE\_BUF\}$, just as every
 51269 application should be prepared to handle partial writes on other kinds of file descriptors.

51270 The intent of forcing writing at least one byte if any can be written is to assure that each write
 51271 makes progress if there is any room in the pipe. If the pipe is empty, $\{PIPE\_BUF\}$ bytes must be
 51272 written; if not, at least some progress must have been made.

51273 Where this volume of IEEE Std 1003.1-2001 requires -1 to be returned and $errno$ set to
 51274 $[EAGAIN]$, most historical implementations return zero (with the O_NDELAY flag set, which is
 51275 the historical predecessor of $O\_NONBLOCK$, but is not itself in this volume of
 51276 IEEE Std 1003.1-2001). The error indications in this volume of IEEE Std 1003.1-2001 were chosen
 51277 so that an application can distinguish these cases from end-of-file. While $write()$ cannot receive
 51278 an indication of end-of-file, $read()$ can, and the two functions have similar return values. Also,
 51279 some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that
 51280 the reader should get an end-of-file indication; for those systems, a return value of zero from
 51281 $write()$ indicates a successful write of an end-of-file indication.

51282 Implementations are allowed, but not required, to perform error checking for *write()* requests of
51283 zero bytes.

51284 The concept of a {PIPE_MAX} limit (indicating the maximum number of bytes that can be
51285 written to a pipe in a single operation) was considered, but rejected, because this concept would
51286 unnecessarily limit application writing.

51287 See also the discussion of O_NONBLOCK in *read()*.

51288 Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be
51289 proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the
51290 calls are made by different processes. A similar requirement applies to multiple write operations
51291 to the same file position. This is needed to guarantee the propagation of data from *write()* calls
51292 to subsequent *read()* calls. This requirement is particularly significant for networked file
51293 systems, where some caching schemes violate these semantics.

51294 Note that this is specified in terms of *read()* and *write()*. The XSI extensions *readv()* and *writv()*
51295 also obey these semantics. A new “high-performance” write analog that did not follow these
51296 serialization requirements would also be permitted by this wording. This volume of
51297 IEEE Std 1003.1-2001 is also silent about any effects of application-level caching (such as that
51298 done by *stdio*).

51299 This volume of IEEE Std 1003.1-2001 does not specify the value of the file offset after an error is
51300 returned; there are too many cases. For programming errors, such as [EBADF], the concept is
51301 meaningless since no file is involved. For errors that are detected immediately, such as
51302 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,
51303 an updated value would be very useful and is the behavior of many implementations.

51304 This volume of IEEE Std 1003.1-2001 does not specify behavior of concurrent writes to a file from
51305 multiple processes. Applications should use some form of concurrency control.

51306 **FUTURE DIRECTIONS**

51307 None.

51308 **SEE ALSO**

51309 *chmod()*, *creat()*, *dup()*, *fcntl()*, *getrlimit()*, *lseek()*, *open()*, *pipe()*, *ulimit()*, *writv()*, the Base
51310 Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <stropts.h>, <sys/uio.h>, <unistd.h>

51311 **CHANGE HISTORY**

51312 First released in Issue 1. Derived from Issue 1 of the SVID.

51313 **Issue 5**

51314 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
51315 Threads Extension.

51316 Large File Summit extensions are added.

51317 The *pwrite()* function is added.

51318 **Issue 6**

51319 The DESCRIPTION states that the *write()* function does not block the thread. Previously this
51320 said “process” rather than “thread”.

51321 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are
51322 marked as part of the XSI STREAMS Option Group.

51323 The following new requirements on POSIX implementations derive from alignment with the
51324 Single UNIX Specification:

- 51325 • The DESCRIPTION now states that if `write()` is interrupted by a signal after it has
51326 successfully written some data, it returns the number of bytes written. In the POSIX.1-1988
51327 standard, it was optional whether `write()` returned the number of bytes written, or whether it
51328 returned `-1` with `errno` set to `[EINTR]`. This is a FIPS requirement.
- 51329 • The following changes are made to support large files:
- 51330 — For regular files, no data transfer occurs past the offset maximum established in the open
51331 file description associated with the *files*.
- 51332 — A second `[EFBIG]` error condition is added.
- 51333 • The `[EIO]` error condition is added.
- 51334 • The `[EPIPE]` error condition is added for when a pipe has only one end open.
- 51335 • The `[ENXIO]` optional error condition is added.
- 51336 Text referring to sockets is added to the DESCRIPTION.
- 51337 The following changes were made to align with the IEEE P1003.1a draft standard:
- 51338 • The effect of reading zero bytes is clarified.
- 51339 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
51340 `write()` results are unspecified for typed memory objects.
- 51341 The following error conditions are added for operations on sockets: `[EAGAIN]`,
51342 `[EWOULDBLOCK]`, `[ECONNRESET]`, `[ENOTCONN]`, and `[EPIPE]`.
- 51343 The `[EIO]` error is changed to “may fail”.
- 51344 The `[ENOBUFFS]` error is added for sockets.
- 51345 The following error conditions are added for operations on sockets: `[EACCES]`, `[ENETDOWN]`,
51346 and `[ENETUNREACH]`.
- 51347 The `writenv()` function is split out into a separate reference page.

51348 **NAME**

51349 writev — write a vector

51350 **SYNOPSIS**

51351 xSI #include <sys/uio.h>

51352 ssize_t writev(int *fildev*, const struct iovec **iov*, int *iovcnt*);

51353

51354 **DESCRIPTION**

51355 The *writev()* function shall be equivalent to *write()*, except as described below. The *writev()*
 51356 function shall gather output data from the *iovcnt* buffers specified by the members of the *iov*
 51357 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*−1]. The *iovcnt* argument is valid if greater than 0 and less than
 51358 or equal to {IOV_MAX}, as defined in <limits.h>.

51359 Each *iovec* entry specifies the base address and length of an area in memory from which data
 51360 should be written. The *writev()* function shall always write a complete area before proceeding to
 51361 the next.

51362 If *fildev* refers to a regular file and all of the *iov_len* members in the array pointed to by *iov* are 0,
 51363 *writev()* shall return 0 and have no other effect. For other file types, the behavior is unspecified.

51364 If the sum of the *iov_len* values is greater than {SSIZE_MAX}, the operation shall fail and no data
 51365 shall be transferred.

51366 **RETURN VALUE**

51367 Upon successful completion, *writev()* shall return the number of bytes actually written.
 51368 Otherwise, it shall return a value of −1, the file-pointer shall remain unchanged, and *errno* shall
 51369 be set to indicate an error.

51370 **ERRORS**51371 Refer to *write()*.51372 In addition, the *writev()* function shall fail if:

51373 [EINVAL] The sum of the *iov_len* values in the *iov* array would overflow an **ssize_t**.

51374 The *writev()* function may fail and set *errno* to:

51375 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

51376 **EXAMPLES**51377 **Writing Data from an Array**

51378 The following example writes data from the buffers specified by members of the *iov* array to the
 51379 file associated with the file descriptor *fd*.

```
51380       #include <sys/types.h>
51381       #include <sys/uio.h>
51382       #include <unistd.h>
51383       ...
51384       ssize_t bytes_written;
51385       int fd;
51386       char *buf0 = "short string\n";
51387       char *buf1 = "This is a longer string\n";
51388       char *buf2 = "This is the longest string in this example\n";
51389       int iovcnt;
51390       struct iovec iov[3];
```

```
51391     iov[0].iov_base = buf0;
51392     iov[0].iov_len = strlen(buf0);
51393     iov[1].iov_base = buf1;
51394     iov[1].iov_len = strlen(buf1);
51395     iov[2].iov_base = buf2;
51396     iov[2].iov_len = strlen(buf2);
51397     ...
51398     iovcnt = sizeof(iov) / sizeof(struct iovec);

51399     bytes_written = writev(fd, iov, iovcnt);
51400     ...
```

51401 APPLICATION USAGE

51402 None.

51403 RATIONALE

51404 Refer to *write()*.

51405 FUTURE DIRECTIONS

51406 None.

51407 SEE ALSO

51408 *readv()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**limits.h**>, <**sys/uio.h**>

51409 CHANGE HISTORY

51410 First released in Issue 4, Version 2.

51411 Issue 6

51412 Split out from the *write()* reference page.

51413 **NAME**

51414 wscanf — convert formatted wide-character input

51415 **SYNOPSIS**

51416 #include <stdio.h>

51417 #include <wchar.h>

51418 int wscanf(const wchar_t *restrict *format*, ...);51419 **DESCRIPTION**51420 Refer to *fwscanf()*.

51421 **NAME**

51422 y0, y1, yn — Bessel functions of the second kind

51423 **SYNOPSIS**

```

51424 xSI      #include <math.h>

51425          double y0(double x);
51426          double y1(double x);
51427          double yn(int n, double x);
51428

```

51429 **DESCRIPTION**

51430 The *y0()*, *y1()*, and *yn()* functions shall compute Bessel functions of *x* of the second kind of
 51431 orders 0, 1, and *n*, respectively.

51432 An application wishing to check for error situations should set *errno* to zero and call
 51433 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 51434 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 51435 zero, an error has occurred.

51436 **RETURN VALUE**

51437 Upon successful completion, these functions shall return the relevant Bessel value of *x* of the
 51438 second kind.

51439 If *x* is NaN, NaN shall be returned.

51440 If the *x* argument to these functions is negative, *-HUGE_VAL* or NaN shall be returned, and a
 51441 domain error may occur.

51442 If *x* is 0.0, *-HUGE_VAL* shall be returned and a range error may occur.

51443 If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

51444 If the correct result would cause overflow, *-HUGE_VAL* or 0.0 shall be returned and a range
 51445 error may occur.

51446 **ERRORS**

51447 These functions may fail if:

51448	Domain Error	The value of <i>x</i> is negative.
51449		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
51450		then <i>errno</i> shall be set to [EDOM]. If the integer expression (<i>math_errhandling</i>
51451		& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
51452		shall be raised.
51453	Range Error	The value of <i>x</i> is 0.0, or the correct result would cause overflow.
51454		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
51455		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
51456		(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
51457		floating-point exception shall be raised.
51458	Range Error	The value of <i>x</i> is too large in magnitude, or the correct result would cause
51459		underflow.
51460		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
51461		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
51462		(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
51463		floating-point exception shall be raised.

51464 **EXAMPLES**

51465 None.

51466 **APPLICATION USAGE**

51467 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
51468 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

51469 **RATIONALE**

51470 None.

51471 **FUTURE DIRECTIONS**

51472 None.

51473 **SEE ALSO**

51474 *feclearexcept()*, *fetestexcept()*, *isnan()*, *j0()*, the Base Definitions volume of IEEE Std 1003.1-2001,
51475 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

51476 **CHANGE HISTORY**

51477 First released in Issue 1. Derived from Issue 1 of the SVID.

51478 **Issue 5**

51479 The DESCRIPTION is updated to indicate how an application should check for an error. This
51480 text was previously published in the APPLICATION USAGE section.

51481 **Issue 6**

51482 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

51483 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
51484 with the ISO / IEC 9899:1999 standard.

Index

<code>_CS_PATH</code>	214	<code>_POSIX2 constants</code>	
<code>_CS_XBS5_ILP32_OFF32_CFLAGS</code>	214	in <code>sysconf</code>	1473
<code>_CS_XBS5_ILP32_OFF32_LDFLAGS</code>	214	<code>_POSIX2_CHAR_TERM</code>	1475
<code>_CS_XBS5_ILP32_OFF32_LIBS</code>	214	<code>_POSIX2_C_BIND</code>	1475
<code>_CS_XBS5_ILP32_OFFBIG_CFLAGS</code>	214	<code>_POSIX2_C_DEV</code>	1475
<code>_CS_XBS5_ILP32_OFFBIG_LDFLAGS</code>	214	<code>_POSIX2_C_VERSION</code>	1475
<code>_CS_XBS5_ILP32_OFFBIG_LIBS</code>	214	<code>_POSIX2_FORT_DEV</code>	1475
<code>_CS_XBS5_LP64_OFF64_CFLAGS</code>	214	<code>_POSIX2_FORT_RUN</code>	1475
<code>_CS_XBS5_LP64_OFF64_LDFLAGS</code>	214	<code>_POSIX2_LOCALEDEF</code>	1475
<code>_CS_XBS5_LP64_OFF64_LIBS</code>	214	<code>_POSIX2_PBS</code>	1475
<code>_CS_XBS5_LPBIG_OFFBIG_CFLAGS</code>	214	<code>_POSIX2_PBS_ACCOUNTING</code>	1475
<code>_CS_XBS5_LPBIG_OFFBIG_LDFLAGS</code>	214	<code>_POSIX2_PBS_LOCATE</code>	1475
<code>_CS_XBS5_LPBIG_OFFBIG_LIBS</code>	214	<code>_POSIX2_PBS_MESSAGE</code>	1475
<code>_exit</code>	85, 307, 1592	<code>_POSIX2_PBS_TRACK</code>	1475
<code>_Exit()</code>	307, 85	<code>_POSIX2_SW_DEV</code>	1475
<code>_FILE</code>	128	<code>_POSIX2_UPE</code>	1475
<code>_IOFBF</code>	1285, 1327	<code>_POSIX2_VERSION</code>	1475
<code>_IOLBF</code>	376, 1327	<code>_POSIX</code>	15, 17
<code>_IONBF</code>	1285, 1327	<code>_POSIX_ADVISORY_INFO</code>	1473
<code>_LINE</code>	128	<code>_POSIX_ASYNCHRONOUS_IO</code>	1473
<code>_longjmp()</code>	86	<code>_POSIX_ASYNC_IO</code>	399
<code>_LVL</code>	17	<code>_POSIX_BARRIERS</code>	1473
<code>_MAX</code>	16	<code>_POSIX_CHOWN_RESTRICTED</code>	190, 399, 401
<code>_MIN</code>	16	<code>_POSIX_CLOCK_SELECTION</code>	1473
<code>_PC constants</code>		<code>_POSIX_CPUTIME</code>	1473
used in <code>pathconf</code>	399	<code>_POSIX_C_SOURCE</code>	14
<code>_PC_ALLOC_SIZE_MIN</code>	399	<code>_POSIX_FILE_LOCKING</code>	1474
<code>_PC_ASYNC_IO</code>	399	<code>_POSIX_FSYNC</code>	1474
<code>_PC_CHOWN_RESTRICTED</code>	399	<code>_POSIX_JOB_CONTROL</code>	1474
<code>_PC_FILESIZEBITS</code>	399	<code>_POSIX_MAPPED_FILES</code>	1474
<code>_PC_LINK_MAX</code>	399	<code>_POSIX_MEMLOCK</code>	1474
<code>_PC_MAX_CANON</code>	399	<code>_POSIX_MEMLOCK_RANGE</code>	1474
<code>_PC_MAX_INPUT</code>	399	<code>_POSIX_MEMORY_PROTECTION</code>	1474
<code>_PC_NAME_MAX</code>	399	<code>_POSIX_MESSAGE_PASSING</code>	1474
<code>_PC_NO_TRUNC</code>	399	<code>_POSIX_MONOTONIC_CLOCK</code>	1474
<code>_PC_PATH_MAX</code>	399	<code>_POSIX_MULTI_PROCESS</code>	1474
<code>_PC_PIPE_BUF</code>	399	<code>_POSIX_NO_TRUNC</code>	399
<code>_PC_PRIO_IO</code>	399	<code>_POSIX_OPEN_MAX</code>	556
<code>_PC_REC_INCR_XFER_SIZE</code>	399	<code>_POSIX_PRIORITIZED_IO</code>	42, 1474
<code>_PC_REC_MAX_XFER_SIZE</code>	399	<code>_POSIX_PRIORITY_SCHEDULING</code>	42, 1474
<code>_PC_REC_MIN_XFER_SIZE</code>	399	<code>_POSIX_PRIO_IO</code>	399
<code>_PC_REC_XFER_ALIGN</code>	399	<code>_POSIX_READER_WRITER_LOCKS</code>	1474
<code>_PC_SYMLINK_MAX</code>	399	<code>_POSIX_REALTIME_SIGNALS</code>	1474
<code>_PC_SYNC_IO</code>	399	<code>_POSIX_REGEX</code>	1474
<code>_PC_VDISABLE</code>	399	<code>_POSIX_SAVED_IDS</code>	1474
		<code>_POSIX_SEMAPHORES</code>	1474

_POSIX_SHARED_MEMORY_OBJECTS	1474	_SC_2_PBS_MESSAGE	1475
_POSIX_SHELL	1474	_SC_2_PBS_TRACK	1475
_POSIX_SOURCE	14	_SC_2_SW_DEV	1475
_POSIX_SPAWN	1474	_SC_2_UPE	1475
_POSIX_SPIN_LOCKS	1474	_SC_2_VERSION	863, 1475
_POSIX_SPORADIC_SERVER	1474	_SC_ADVISORY_INFO	1473
_POSIX_SYNCHRONIZED_IO	1474	_SC_AIO_LISTIO_MAX	1473
_POSIX_SYNC_IO	399	_SC_AIO_MAX	1473
_POSIX_THREADS	241, 1474, 1533	_SC_AIO_PRIO_DELTA_MAX	1473
_POSIX_THREAD_ATTR_STACKADDR	1474	_SC_ARG_MAX	1473
_POSIX_THREAD_ATTR_STACKSIZE	1474	_SC_ASYNCHRONOUS_IO	1473
_POSIX_THREAD_CPUTIME	1474	_SC_ATEXIT_MAX	1473
_POSIX_THREAD_PRIORITY_SCHEDULING	1474	_SC_BARRIERS	1473
_POSIX_THREAD_PRIO_INHERIT	1474	_SC_BC_BASE_MAX	1473
_POSIX_THREAD_PRIO_PROTECT	1474	_SC_BC_DIM_MAX	1473
_POSIX_THREAD_PROCESS_SHARED	1092	_SC_BC_SCALE_MAX	1473
.....	1474	_SC_BC_STRING_MAX	1473
_POSIX_THREAD_SAFE_FUNCTIONS	241	_SC_CHILD_MAX	1473
.....	1474, 1533	_SC_CLK_TCK	1473, 1529
_POSIX_THREAD_SPARADIC_SERVER	1474	_SC_CLOCK_SELECTION	1473
_POSIX_TIMEOUTS	1474	_SC_COLL_WEIGHTS_MAX	1473
_POSIX_TIMERS	1474	_SC_CPUTIME	1473
_POSIX_TRACE	1474	_SC_DELAYTIMER_MAX	1473
_POSIX_TRACE_EVENT_FILTER	1474	_SC_EXPR_NEST_MAX	1473
_POSIX_TRACE_EVENT_NAME_MAX	939, 941	_SC_FILE_LOCKING	1474
_POSIX_TRACE_INHERIT	1474	_SC_FSYNC	1474
_POSIX_TRACE_LOG	1474	_SC_GETGR_R_SIZE_MAX	507, 510, 1473
_POSIX_TRACE_SYS_MAX	936	_SC_GETPW_R_SIZE_MAX	548, 551, 1473
_POSIX_TRACE_USER_EVENT_MAX	941	_SC_IOV_MAX	1473
_POSIX_TYPED_MEMORY_OBJECTS	1474	_SC_JOB_CONTROL	1474
_POSIX_V6_ILP32_OFF32	1474	_SC_LINE_MAX	1473
_POSIX_V6_ILP32_OFFBIG	1474	_SC_LOGIN_NAME_MAX	1473
_POSIX_V6_LP64_OFF64	1474	_SC_MEMLOCK	1474
_POSIX_V6_LPBIG_OFFBIG	1475	_SC_MEMLOCK_RANGE	1474
_POSIX_VDISABLE	399	_SC_MEMORY_PROTECTION	1474
_POSIX_VERSION	1474, 1557	_SC_MESSAGE_PASSING	1474
_PROCESS	17	_SC_MONOTONIC_CLOCK	1474
_PTHREAD_THREADS_MAX	1066	_SC_MQ_OPEN_MAX	1473
_REGEX_VERSION	1475	_SC_MQ_PRIO_MAX	1473
_SC constants		_SC_MULTI_PROCESS	1474
in sysconf	1473	_SC_NGROUPS_MAX	1473
_SC_2_CHAR_TERM	1475	_SC_OPEN_MAX	1473
_SC_2_C_BIND	1475	_SC_PAGESIZE	776, 868, 1475
_SC_2_C_DEV	1475	_SC_PAGE_SIZE	776, 1475
_SC_2_C_VERSION	1475	_SC_PRIORITIZED_IO	1474
_SC_2_FORT_DEV	1475	_SC_PRIORITY_SCHEDULING	1474
_SC_2_FORT_RUN	1475	_SC_READER_WRITER_LOCKS	1474
_SC_2_LOCALEDEF	1475	_SC_REALTIME_SIGNALS	1474
_SC_2_PBS_ACCOUNTING	1475	_SC_REGEX	1474
_SC_2_PBS_LOCATE	1475	_SC_REGEX_VERSION	1475
		_SC_RE_DUP_MAX	1475

_SC_RTSIG_MAX	1475	_SC_XOPEN_REALTIME	1475
_SC_SAVED_IDS	1474	_SC_XOPEN_REALTIME_THREADS	1475
_SC_SEMAPHORES	1474	_SC_XOPEN_SHM	1475
_SC_SEM_NSEMS_MAX	1475	_SC_XOPEN_UNIX	1475
_SC_SEM_VALUE_MAX	1475	_SC_XOPEN_VERSION	1475
_SC_SHARED_MEMORY_OBJECTS	1474	_SC_XOPEN_XCU_VERSION	1476
_SC_SHELL	1474	_setjmp	86
_SC_SIGQUEUE_MAX	1475	_t	17
_SC_SPAWN	1474	_TIME	17
_SC_SPIN_LOCKS	1474	_tolower()	88
_SC_SPORADIC_SERVER	1474	_toupper()	89
_SC_STREAM_MAX	1475	_XBS5_ILP32_OFF32	1475
_SC_SYMLINK_MAX	1475	_XBS5_ILP32_OFFBIG	1475
_SC_SYNCHRONIZED_IO	1474	_XBS5_LP64_OFF64	1475
_SC_THREADS	1474	_XBS5_LPBIG_OFFBIG	1475
_SC_THREAD_ATTR_STACKADDR	1474	_XOPEN_CRYPT	1475
_SC_THREAD_ATTR_STACKSIZE	1474	_XOPEN_ENH_I18N	1475
_SC_THREAD_CPUTIME	1474	_XOPEN_LEGACY	1475
_SC_THREAD_DESTRUCTOR_ITERATIONS	1475	_XOPEN_REALTIME	337, 1475
_SC_THREAD_KEYS_MAX	1475	_XOPEN_REALTIME_THREADS	1475
_SC_THREAD_PRIORITY_SCHEDULING	1474	_XOPEN_SHM	1475
_SC_THREAD_PRIO_INHERIT	1474	_XOPEN_SOURCE	14
_SC_THREAD_PRIO_PROTECT	1474	_XOPEN_UNIX	1475
_SC_THREAD_PROCESS_SHARED	1474	_XOPEN_VERSION	1475
_SC_THREAD_SAFE_FUNCTIONS	1474	_XOPEN_XCU_VERSION	1476
_SC_THREAD_SPORADIC_SERVER	1474	a64l()	90
_SC_THREAD_STACK_MIN	1475	ABDAY_1	834
_SC_THREAD_THREADS_MAX	1475	abort()	92
_SC_TIMEOUTS	1474	abs()	94
_SC_TIMERS	1474	accept()	95
_SC_TIMER_MAX	1475	access()	97
_SC_TRACE	1474	acos()	100
_SC_TRACE_EVENT_FILTER	1474	acosf	100
_SC_TRACE_INHERIT	1474	acosh()	102
_SC_TRACE_LOG	1474	acoshf	102
_SC_TTY_NAME_MAX	1475	acoshl	102
_SC_TYPED_MEMORY_OBJECTS	1474	acosl()	100, 104
_SC_TZNAME_MAX	1475	ACTION	584
_SC_V6_ILP32_OFF32	1474	address information	426
_SC_V6_ILP32_OFFBIG	1474	address string	426
_SC_V6_LP64_OFF64	1474	addrinfo structure	426
_SC_V6_LPBIG_OFFBIG	1475	ADV	3
_SC_VERSION	1474	ADVANCED REALTIME	196, 200, 800-801, 864
_SC_XBS5_ILP32_OFF32	1475	866, 868, 870, 872, 875, 883, 886	
_SC_XBS5_ILP32_OFFBIG	1475	888-890, 892, 894, 896, 898, 900	
_SC_XBS5_LP64_OFF64	1475	902, 904-911, 964, 966, 1043, 1088, 1259	
_SC_XBS5_LPBIG_OFFBIG	1475	ADVANCED REALTIME THREADS	1012, 1014
_SC_XOPEN_CRYPT	1475	1016, 1018, 1020-1021, 1060, 1141	
_SC_XOPEN_ENH_I18N	1475	1143, 1145	
_SC_XOPEN_LEGACY	1475	AF	18
		AIO	3

AIO_	16	atol()	141
aio_	16	atoll	141
AIO_ALLDONE	105	attributes, clock-resolution	76, 914
aio_cancel()	105	attributes, creation-time	76, 914
AIO_CANCELED	105	attributes, generation-version	75, 914
aio_error()	107	attributes, inheritance	76, 916
aio_fsync()	108	attributes, log-full-policy	74, 76, 916, 919
AIO_LISTIO_MAX	687, 1473	attributes, log-max-size	76, 917, 919
AIO_MAX	687, 1473	attributes, max-data-size	76, 919-920
AIO_NOTCANCELED	105	attributes, stream-full-policy	72-73, 76, 917
AIO_PRIO_DELTA_MAX	42, 1473	attributes, stream-min-size	76, 920
aio_read()	110	attributes, trace-name	76, 914
aio_return()	113	attributes, truncation-status	939
aio_suspend()	114	background	1305
aio_write()	116	background process	1504
AI_ALL	427	BAR	3
AI_CANONNAME	427	basename()	142
AI_INET6	427	baud rate functions	180
AI_NUMERICHOST	427	bcmp()	144
AI_NUMERICSERV	427	bcopy()	145
AI_PASSIVE	427	BC_constants	
AI_V4MAPPED	427	in sysconf	1473
alarm()	119	BC_BASE_MAX	1473
anycast	67	BC_DIM_MAX	1473
ANYMARK	620	BC_SCALE_MAX	1473
appropriate privileges	98, 400	BC_STRING_MAX	1473
argc	302	BE	4
ARG_MAX	22, 297, 300, 304, 1473	bind()	146
asctime()	121	bi_	16
asctime_r	121	BOOT_TIME	284-285
asin()	123	broadcasting a condition	1030
asinf	123	BSD	119, 191, 310, 334, 343, 401
asinh()	125		539, 669, 756, 823, 1176, 1216
asinhf	125		1224, 1305, 1348, 1372, 1496, 1519
asinhf	125		1557, 1592
asinhl	125	bsd_signal()	148
asinl()	123, 127	bsearch()	150
assert()	128	btowc()	153
async-signal-safe	979	buffer cache	452
atan()	129	BUFSIZ	1285
atan2()	131	BUS_	18
atan2f	131	byte-oriented stream	36
atan2l	131	byte-stream mode	1173
atanf()	129, 133	bzero()	154
atanh()	134	cabs()	155
atanhf	134	cabsf	155
atanhl	134	cabsl	155
atanl()	129, 136	cacos()	156
atexit()	137	cacosf	156
ATEXIT_MAX	137, 1473	cacosh()	157
atof()	138	cacoshf	157
atoi()	139		

cacoshl.....	157
cacosl()	156, 158
calloc()	159
can.....	1
cancel-safe.....	1133
cancelability state.....	1067, 1133
cancelability states.....	54
cancelability type	1067, 1133
canceled execution of a thread	1022
canonical name	427
carg().....	161
cargf.....	161
cargl.....	161
casin()	162
casinf.....	162
casinh().....	163
casinhf.....	163
casinhl.....	163
casinl().....	162, 164
catan().....	165
catanf.....	165
catanh()	166
catanhf.....	166
catanhl.....	166
catanl()	165, 167
catclose().....	168
catgets().....	169
catopen().....	171
CBAUD	19
cbrt()	173
cbrtf.....	173
cbrtl.....	173
ccos().....	174
ccosf.....	174
ccosh()	175
ccoshf.....	175
ccoshl.....	175
ccosl()	174, 176
CD.....	4
ceil()	177
ceilf.....	177
ceill.....	177
cexp()	179
cexpf.....	179
cexpl.....	179
cfgetispeed().....	180
cfgetospeed().....	182
cfsetispeed()	183
cfsetospeed()	184
change current working directory	186
change file modes	189
change owner and group of file.....	191
CHAR_MAX	697, 699
chdir().....	185
CHILD_MAX	395, 1473
chmod().....	187
chown()	190
cimag()	193
cimagf.....	193
cimagl.....	193
CLD_	18
clearerr().....	194
clock tick	119, 1477, 1529
clock ticks/second.....	1473
clock().....	195
clock-resolution attribute	76, 914
CLOCKS_PER_SEC	195
CLOCK_	17
clock_.....	17
clock_getcpuclockid()	196
clock_getres().....	197
clock_gettime	197
CLOCK_MONOTONIC	49, 201
clock_nanosleep()	200
CLOCK_PROCESS_CPUTIME_ID.....	49
CLOCK_REALTIME.....	49, 197, 201
.....	823, 1088, 1259, 1521
clock_settime()	197, 203
CLOCK_THREAD_CPUTIME_ID.....	50
clog().....	204
clogf.....	204
clogl.....	204
close a file	207
close()	205
closedir().....	208
closelog().....	210
cmsg_	16
CMSG_.....	18
COLL_WEIGHTS_MAX.....	1473
command interpreter	
portable	1592
compare thread IDs.....	1055
compilation environment	13
condition variable initialization attributes	1041
conforming application.....	1389
conforming application, strictly.....	119, 302
confstr().....	214
conj().....	217
conf	217
conjl.....	217
connect()	218
control data	38

control-normal	1173	ctanl()	238, 240
conversion descriptor.....	297, 302, 590-593	ctermid()	241
conversion specification	404, 435, 468	ctime()	243
.....	477, 1424, 1428, 1441	ctime_r	243
modified	1430	CX.....	4
conversion specifier		c_	17
modified	1442	data key creation	1071
copysign()	221	data messages.....	38
copysignf.....	221	data type	80
copysignl	221	DATMSK.....	495
core	1593	daylight	245 , 1549
core file.....	309	DBL_MANT_DIG	177, 377
cos()	222	DBL_MAX_EXP.....	177, 377
cosf.....	222	DBM	246-247
cosh()	224	dbm_.....	16
coshf	224	DBM.....	18
coshl.....	224	dbm_clearerr().....	246
cosl()	222, 226	dbm_close	246
covert channel	669	dbm_delete	246
cpow()	227	dbm_error	246
cpowf.....	227	dbm_fetch	246
cpowl.....	227	dbm_firstkey.....	246
cproj()	228	DBM_INSERT	246
cprojf	228	dbm_nextkey.....	246
cprojl.....	228	dbm_open	246
CPT	4	DBM_REPLACE	246
creal()	229	dbm_store	246
crealf.....	229	DEAD_PROCESS.....	284-285
creall	229	DEFCHO	19
creat()	230	deferred cancelability	1067
create a per-process timer.....	1522	delay process execution	1388
create an interprocess channel.....	856	DELAUTIMER_MAX.....	1473, 1526
create session and set process group ID.....	1317	dependency ordering.....	259
creation-time attribute	76, 914	descriptive name	426
CRYPT.....	232, 270, 1298	destroying a mutex	1077
crypt()	232	destroying condition variables.....	1034
CS	4	destructor functions.....	1071
csin()	234	detaching a thread.....	1053
csinf	234	diffime().....	250
csinh()	235	DIR	80, 208, 845, 1179, 1181, 1219, 1245, 1512
csinhf.....	235	directive.....	404, 435, 468, 477, 1441
csinhl	235	directory operations.....	846
csinl()	234, 236	dirent structure	846
csqrt()	237	dirname()	251
csqrtf.....	237	div().....	253
csqrtl.....	237	dlclose().....	254
ctan()	238	dlderror().....	256
ctanf.....	238	dlopen().....	258
ctanh()	239	dlsym()	261
ctanhf.....	239	dot.....	846, 1216
ctanhl.....	239	dot-dot	846, 1216

drand48()	263	encrypt()	270
dup()	265	endgrent()	272
dup2	265	endhostent()	274
dynamic package initialization	1109	endnetent()	276
d_	16	endprotoent()	278
E2BIG	22	endpwent()	280
EACCES	22	endservent()	282
EADDRINUSE	22	endutxent()	284
EADDRNOTAVAIL	22	ENETDOWN	25
EAFNOSUPPORT	22	ENETRESET	25
EAGAIN	22, 28	ENETUNREACH	25
EALREADY	22	ENFILE	25
EBADF	22	ENOBUFFS	25
EBADMSG	22	ENODATA	25
EBUSY	23	ENODEV	25
ECANCELED	23	ENOENT	25
ECHILD	23	ENOEXEC	25
ECHOCTL	19	ENOLCK	25
ECHOKE	19	ENOLINK	25
ECHOPRT	19	ENOMEM	25
ECONNABORTED	23	ENOMSG	25
ECONNREFUSED	23	ENOPROTOOPT	25
ECONNRESET	23	ENOSPC	26
ecvt()	268	ENOSR	26
EDEADLK	23	ENOSTR	26
EDESTADDRREQ	23	ENOSYS	26
EDOM	23	ENOTCONN	26
EDQUOT	23	ENOTDIR	26
EEXIST	23	ENOTEMPTY	26
EFAULT	23	ENOTSOCK	26
EFBIG	23	ENOTSUP	26
effective group ID	191, 304, 513	ENOTTY	26
effective user ID	98, 304, 669	ENTRY	584
EHOSTUNREACH	23	environ	287, 303
EIDRM	23	envp	303
Eighth Edition UNIX	1661	ENXIO	26
EILSEQ	24, 37	EOPNOTSUPP	26
EINPROGRESS	24, 42	EOVERFLOW	26
EINTR	24, 57, 933	EPERM	26
EINVAL	24, 920, 933	EPIPE	26
EIO	24	EPROTO	27
EISCONN	24	EPROTONOSUPPORT	27
EISDIR	24	EPROTOTYPE	27
ELOOP	24	erand48()	263, 288
ELSIZE	726	ERANGE	27
EMFILE	24	erf()	289
EMLINK	24	erfc()	291
EMPTY	285	erfcf	291
EMSGSIZE	24	erfcl	291
EMULTIHOP	24	erff()	289, 293
ENAMETOOLONG	25	erfl	289, 293

EROFS	27	fchdir()	322
errno	294	fchmod()	323
error descriptions	483	fchown()	325
error numbers	21	fclose()	327
additional	28	fcntl()	329
ESPIPE	27	fcvt()	268, 336
ESRCH	27	FD	4
EST5EDT	1549	fdatasync()	337
ESTALE	27	fdetach()	338
ETIME	27	fdim()	340
ETIMEDOUT	27	fdimf	340
ETXTBSY	27	fdiml	340
EWOLDBLOCK	27	fdopen()	342
examine and change blocked signals	1140	fds_	16
examine and change signal action	1348	FD_	16, 18
EXDEV	28	fd_	16
exec	296	FD_CLOEXEC	35, 171, 297, 329, 593
of shell scripts	303		838, 846, 856, 876, 883, 1329
exec family ...98, 207, 333, 374, 397, 979, 1306, 1592		FD_CLR()	84, 974
execl	296	FD_ISSET	84, 974
execle	296	FD_SET	84, 974
execlp	296	FD_ZERO	84, 974
execute a file	302	feature test macro	13, 488
execution time monitoring	49	_POSIX_C_SOURCE	14
execv	296	_XOPEN_SOURCE	14
execve	296	feclearexcept()	344
execvp	296	fegetenv()	345
exit()	307	fegetexceptflag()	346
EXIT_FAILURE	307	fegetround()	347
EXIT_SUCCESS	307, 310	feholdexcept()	349
exp()	312	feof()	350
exp2()	314	feraiseexcept()	351
exp2f	314	ferror()	352
exp2l	314	fesetenv()	345, 353
expf	312	fesetexceptflag()	346, 354
expl	312	fesetround()	347, 355
expm1()	316	fetestexcept()	356
expm1f	316	feupdateenv()	358
expm1l	316	fflush()	360
EXPR_NEST_MAX	1473	ffs()	363
EXTA	19	fgetc()	364
EXTB	19	fgetpos()	366
extension		fgets()	368
CX	4	fgetwc()	370
OH	7	fgetws()	372
XSI	11	FIFO	758-759, 842, 1660
extensions to setlocale	1300	FILE	80, 194, 327, 342, 350, 352
fabs()	318		360, 364, 366, 368, 370, 372
fabsf	318		374-375, 390, 404, 416, 418, 420
fabsl	318		422-423, 430, 435, 442, 445, 454
fattach()	319		466-468, 475, 477, 486-487, 573

.....	852, 862, 1148-1149, 1161, 1218
.....	1285, 1327, 1409, 1531, 1559-1560
.....	1577, 1579, 1581, 1583, 1585, 1587
file	
locking	332
file accessibility	98
file control	332
FILE object.....	34
file permission bits	98
file permissions.....	98, 401, 1406
file position indicator.....	34
fileno()	374
FILESIZEBITS.....	399
FIND.....	584
find string token.....	1454
flockfile().....	375
floor()	377
floorf	377
floorl	377
FLT_RADIX.....	716
FLT_ROUNDS.....	379
FLUSH.....	18
FLUSHO	19
FLUSHR.....	614
FLUSHRW	614
FLUSHW	614
fma()	379
fmaf.....	379
fmal.....	379
fmax()	381
fmaxf	381
fmaxl.....	381
fmin()	382
fminf	382
fminl	382
FMNAMESZ	613
fmod().....	383
fmodf.....	383
fmodl.....	383
fmtmsg()	385
fnmatch()	388
FNM_	18
FNM_NOESCAPE.....	388
FNM_NOMATCH.....	388
FNM_PATHNAME.....	388
FNM_PERIOD.....	388
fopen()	390
FOPEN_MAX.....	342, 391, 862, 1531
foreground	1305
fork handler	980
fork().....	394
forkall	397
format of entries.....	11
fpathconf().....	399
fpclassify()	403
FPE_.....	18
fprintf().....	404
fputc().....	416
fputs().....	418
fputwc()	420
fputws().....	422
FQDN	528
FR	4
fread().....	423
free().....	425
freeaddrinfo().....	426
freopen()	430
frexp().....	433
frexpf	433
frexpl	433
FSC	5
fscanf().....	435
fseek()	442
fseeko	442
fsetpos()	445
fstat().....	447
fstatvfs().....	449
fsync().....	452
ftell()	454
ftello.....	454
ftime().....	456
ftok()	458
ftruncate().....	460
ftrylockfile()	375, 462
FTW	18, 829-830
ftw()	463
FTW_CHDIR.....	829
FTW_D	463, 829
FTW_DEPTH.....	829
FTW_DNR	463, 829-830
FTW_DP	829
FTW_F	463, 829
FTW_MOUNT	829
FTW_NS	463, 829-830
FTW_PHYS.....	829
FTW_SL	463, 829
FTW_SLN	829
fully-qualified domain name	528
functions	13
implementation.....	13
use.....	13
funlockfile().....	375, 466

<code>fwide()</code>	467	<code>getgrnam()</code>	510
<code>fwprintf()</code>	468	<code>getgrnam_r</code>	510
<code>fwrite()</code>	475	<code>getgroups()</code>	512
<code>fwscanf()</code>	477	<code>gethostbyaddr()</code>	514
<code>f_</code>	16	<code>gethostbyname</code>	514
<code>F_</code>	18	<code>gethostent()</code>	274, 516
<code>F_DUPFD</code>	265, 329, 331-332	<code>gethostid()</code>	517
<code>F_GETFD</code>	329, 331-332	<code>gethostname()</code>	518
<code>F_GETFL</code>	329, 331-332	<code>getitimer()</code>	519
<code>F_GETLK</code>	330-332	<code>getlogin()</code>	521
<code>F_GETOWN</code>	329, 331	<code>getlogin_r</code>	521
<code>F_LOCK</code>	705	<code>getmsg()</code>	524
<code>F_RDLCK</code>	332	<code>getnameinfo()</code>	528
<code>F_SETFD</code>	329, 331-333	<code>GETNCNT</code>	1266-1267
<code>F_SETFL</code>	329, 331-332	<code>getnetbyaddr()</code>	276, 531
<code>F_SETLK</code>	330-332	<code>getnetbyname</code>	276, 531
<code>F_SETLKW</code>	55, 330-332	<code>getnetent</code>	276, 531
<code>F_SETOWN</code>	329, 332	<code>getopt()</code>	532
<code>F_TEST</code>	705	<code>getpeername()</code>	537
<code>F_TLOCK</code>	705	<code>getpgid()</code>	538
<code>F_ULOCK</code>	705	<code>getpgrp()</code>	539
<code>F_UNLCK</code>	330-331	<code>GETPID</code>	1266-1267
<code>F_WRLCK</code>	332	<code>getpid()</code>	540
<code>gai_strerror()</code>	483	<code>getpmsg()</code>	524, 541
<code>gcvt()</code>	268, 484	<code>getppid()</code>	542
generation-version attribute.....	75, 914	<code>getpriority()</code>	543
get configurable pathname variables.....	401	<code>getprotent</code>	546
get configurable system variables.....	1476	<code>getprotobyname()</code>	278, 546
get file status.....	1406	<code>getprotobynumber</code>	278, 546
get process times.....	1529	<code>getprotoent</code>	278
get supplementary group IDs.....	512	<code>getpwent()</code>	280, 547
get system time.....	1519	<code>getpwnam()</code>	548
get thread ID.....	1131	<code>getpwnam_r</code>	548
get user name.....	522	<code>getpwuid()</code>	551
<code>getaddrinfo()</code>	426, 485	<code>getpwuid_r</code>	551
<code>GETALL</code>	1266	<code>getrlimit()</code>	554
<code>getc()</code>	486	<code>getrusage()</code>	557
<code>getchar()</code>	489	<code>gets()</code>	559
<code>getchar_unlocked()</code>	487, 490	<code>getservbyname()</code>	282, 560
<code>getcontext()</code>	491	<code>getservbyport</code>	282, 560
<code>getcwd()</code>	493	<code>getservent</code>	282, 560
<code>getc_unlocked()</code>	487	<code>getsid()</code>	561
<code>getdate()</code>	495	<code>getsockname()</code>	562
<code>getdate_err</code>	495	<code>getsockopt()</code>	563
<code>getegid()</code>	500	<code>getsubopt()</code>	566
<code>getenv()</code>	303, 501	<code>gettimeofday()</code>	570
<code>geteuid()</code>	504	<code>getuid()</code>	571
<code>getgid()</code>	505	<code>getutxent</code>	284
<code>getgrent()</code>	272, 506	<code>getutxent()</code>	572
<code>getgrgid()</code>	507	<code>getutxid</code>	284, 572
<code>getgrgid_r</code>	507	<code>getutxline</code>	284, 572

GETVAL.....	1266-1267	ifc.....	16
getwc().....	573	ifra.....	16
getwchar().....	574	ifru.....	16
getwd().....	494, 575	if.....	16
GETZCNT.....	1266-1267	IF.....	18
glob().....	576	if_freenameindex().....	595
globfree.....	576	if_indextname().....	596
GLOB.....	18	if_nameindex().....	597
GLOB_constants		if_nametoindex().....	598
error returns of glob.....	578	ILL.....	18
used in glob.....	576	ilogb().....	599
GLOB_ABORTED.....	578	ilogbf.....	599
GLOB_APPEND.....	576-577	ilogbl.....	599
GLOB_DOOFFS.....	576-577	imaxabs().....	601
GLOB_ERR.....	576, 578	imaxdiv().....	602
GLOB_MARK.....	577	implementation-defined.....	1
GLOB_NOCHECK.....	577-578	IMPLINK.....	18
GLOB_NOESCAPE.....	577	in6.....	16
GLOB_NOMATCH.....	578	IN6.....	18
GLOB_NOSORT.....	577	INADDR.....	18
GLOB_NOSPACE.....	578	index().....	603
gl.....	16	inet.....	16
GMT0.....	1549	inet_addr().....	604
gmtime().....	580	inet_ntoa.....	604
gmtime_r.....	580	inet_ntop().....	606
grantpt().....	582	inet_pton.....	606
granularity of clock.....	456	Inf.....	123
HALT.....	386	INF.....	407, 471
hcreate().....	584	INFINITY.....	407, 471
hdestroy.....	584	INFO.....	386
high resolution sleep.....	823	infu.....	16
host name.....	426	inheritance attribute.....	76, 916
htonl().....	587	init.....	310, 669
htons.....	587	initialize a named semaphore.....	1255
HUGE_VAL.....	134, 177, 224, 312, 314	initialize an unnamed semaphore.....	1252
.....	316, 340, 377, 588, 677, 681, 708	initializing a mutex.....	1077
.....	712, 714, 825, 827, 969, 1221, 1226	initializing condition variables.....	1034
.....	1230, 1385, 1449, 1485, 1516, 1619	initstate().....	608
HUGE_VALF.....	377, 1221, 1449	INIT_PROCESS.....	284-285
HUGE_VALL.....	377, 1221, 1449	input and output rationale.....	1175
hypot().....	588	insque().....	610
hypotf.....	588	international environment.....	1300
hypotl.....	588	Internet Protocols.....	66
h.....	16	INT_MAX.....	599
h_errno.....	583	INT_MIN.....	94
iconv().....	590	in.....	16
iconv_close().....	592	IN.....	18
iconv_open().....	593	ioctl().....	613
ic.....	16	iov.....	17
IEEE Std 754-1985.....	3	IOV.....	18
IEEE Std 854-1987.....	3	IOV_MAX.....	1186, 1473, 1664

IP6	5	isinf()	637
IPC.....	39, 806, 808, 811, 813, 1271, 1275, 1339, 1341	isless()	638
ipc_	16	islessequal()	639
IPC_	18	islessgreater()	640
IPC_ constants		islower()	641
used in semctl	1266	isnan()	643
used in shmctl	1337	isnormal()	644
IPC_CREAT	807, 1269, 1340	ISO C standard	3, 119, 302, 332
IPC_EXCL	807, 1269		488, 728, 1169, 1216, 1300, 1348
IPC_NOWAIT	809-810, 812-813, 1272		1372, 1519
IPC_PRIVATE	807, 1269, 1340	isprint()	645
IPC_RMID	805, 1267, 1337	ispunct()	646
IPC_SET	805, 1267, 1337	isspace()	647
IPC_STAT	805, 1266, 1337	isunordered()	648
IPPORT_	18	isupper()	649
IPPROTO_	18	iswalnum()	650
IPv4	67	iswalpha()	651
IPv4-compatible address	68	iswblank()	652
IPv4-mapped address	68	iswcntrl()	653
IPv6	67	iswctype()	654
compatibility with IPv4	68	iswdigit()	656
interface identification	68	iswgraph()	657
options	69	iswlower()	658
IPv6 address		iswprint()	659
anycast	67	iswpunct()	660
loopback	68	iswspace()	661
multicast	67	iswupper()	662
unicast	67	iswxdigit()	663
unspecified	68	isxdigit()	664
IPV6_	18	ITIMER_PROF	519
IPV6_JOIN_GROUP	69	ITIMER_REAL	519
IPV6_LEAVE_GROUP	69	ITIMER_VIRTUAL	519
IPV6_MULTICAST_HOPS	69	it_	16-17
IPV6_MULTICAST_IF	69	I_	18
IPV6_MULTICAST_LOOP	69	I_ATMARK	619-620
IPV6_UNICAST_HOPS	69	I_CANPUT	620
IPV6_V6ONLY	69	I_CKBAND	620
ip_	16	I_FDINSERT	616
IP_	18	I_FIND	615
isalnum()	625	I_FLUSH	613
isalpha()	626	I_FLUSHBAND	614
iscii()	627	I_GETBAND	620
isastream()	628	I_GETCLTIME	620
isatty()	629	I_GETSIG	615
isblank()	630	I_GRDOPT	616, 1173
iscntrl()	631	I_GWROPT	618
isdigit()	632	I_LINK	621
isfinite()	633	I_LIST	619
isgraph()	634	I_LOOK	613
isgreater()	635	I_NREAD	616
isgreaterequal()	636	I_PEEK	615

I_PLINK.....	622	link().....	683
I_POP.....	613	LINK_MAX.....	24, 399, 683, 1215
I_PUNLINK.....	622	LIO_.....	16
I_PUSH.....	613	lio_.....	16
I_RECVFD.....	22, 619	lio_listio().....	686
I_SENDFD.....	618-619	LIO_NOP.....	686
I_SETCLTIME.....	205, 620	LIO_NOWAIT.....	686
I_SETSIG.....	614-615	LIO_READ.....	686
I_SRDOPT.....	615-616, 1173	LIO_WAIT.....	686
I_STR.....	617	LIO_WRITE.....	686
I_SWROPT.....	618, 1658	list directed I/O.....	688
I_UNLINK.....	621	listen().....	689
j0().....	665	llabs().....	673, 691
j1.....	665	lldiv().....	679, 692
jn.....	665	LLONG_MAX.....	1457, 1625
job control.....	310, 539, 669, 1305, 1317, 1476, 1592	LLONG_MIN.....	1457
rand48().....	263, 667	llrint().....	693
JST-9.....	1549	llrintf.....	693
kill().....	668	llrintl.....	693
killpg().....	671	llround().....	695
l64a().....	90, 672	llroundf.....	695
labs().....	673	llroundl.....	695
LANG.....	171	load ordering.....	259
last close.....	1333	LOBLK.....	19
LASTMARK.....	620	localeconv().....	697
lchown().....	674	localtime().....	702
lcong48().....	263, 676	localtime_r.....	702
LC_ALL.....	297, 699, 834, 1299, 1301	lockf().....	705
LC_COLLATE.....	576-577, 1299-1300, 1416	locking.....	332
.....	1464, 1603, 1637	advisory.....	333
LC_CTYPE.....	153, 654, 736, 738	mandatory.....	333
.....	740, 742-743, 745, 747, 1299-1300	locking and unlocking a mutex.....	1085
.....	1536-1540, 1598, 1614, 1629	log().....	708
.....	1639-1640, 1642-1643	log-full-policy attribute.....	74, 76, 916, 919, 937
LC_MESSAGES.....	171, 1299-1300, 1422	log-max-size attribute.....	76, 917, 919
LC_MONETARY.....	699, 1299-1300, 1425	log10().....	710
LC_NUMERIC.....	268, 405, 435, 468, 477	log10f.....	710
.....	699, 1299-1300, 1425, 1449, 1619	log10l.....	710
LC_TIME.....	496, 834, 1299-1300	log1p().....	712
ldexp().....	677	log1pf.....	712
ldexpf.....	677	log1pl.....	712
ldexpl.....	677	log2().....	714
ldiv().....	679	log2f.....	714
legacy.....	1	log2l.....	714
lfind().....	680 , 726	logb().....	716
lgamma().....	681	logbf.....	716
lgammaf.....	681	logbl.....	716
lgammal.....	681	logf().....	708, 718
LIFO.....	57	login shell.....	302
LINE_MAX.....	1473	LOGIN_NAME_MAX.....	521, 1473
link to a file.....	685	LOGIN_PROCESS.....	284-285

logl.....	708, 718	mbsrtowcs()	743
LOG_.....	18	mbstowcs()	745
LOG_constants in syslog	210	mbtowc()	747
LOG_ALERT	210	MB_CUR_MAX	736, 738, 740, 747, 1598, 1640
LOG_CONS	211	MC1	5
LOG_CRIT	210	MC2	5
LOG_DEBUG	210	MC3	5
LOG_EMERG	210	MCL	16
LOG_ERR	210	MCL_CURRENT	772
LOG_INFO	210	MCL_FUTURE	772
LOG_LOCAL	210	memccpy()	749
LOG_NDELAY	211	memchr()	750
LOG_NOTICE	210	memcmp()	751
LOG_NOWAIT	211	memcpy()	752
LOG_ODELAY	211	MEMLOCK_FUTURE	779
LOG_PID	211	memmove()	753
LOG_USER	210-211	memory management	43
LOG_WARNING	210	memory protection option	778
longjmp()	719	memset()	754
LONG_MAX	1457, 1625	message catalog descriptor	297, 302, 307
LONG_MIN	1457, 1625	message parts	39
lrand48()	263, 721	message priority	38
lrint()	722	high-priority	38
lrintf	722	normal	38
lrintl	722	priority	38
lround()	724	message-discard mode	1173
lroundf	724	message-nondiscard mode	1173
lroundl	724	MET-1MEST	1549
lsearch()	726	MF	5
lseek()	728	MINSIGSTKSZ	1351
lstat()	730	mkdir()	755
l_	16	mkfifo()	758
L_ctermid	241	mknod()	761
l_sysid	333	mkstemp()	764
makecontext()	732	mktemp()	766
malloc()	734	mktime()	768
manipulate signal sets	1354	ML	6
mappings	779	mlock()	770
MAP_	16, 18	mlockall()	772
MAP_FAILED	779	MLR	6
MAP_FIXED	775, 778	mmap()	774
MAP_PRIVATE	394, 775, 779, 783, 815	MM_	18
MAP_SHARED	397, 775-776	MM_APPL	385
max-data-size attribute	76, 919-920	MM_CONSOLE	385
MAX_CANON	399	MM_ERROR	386-387
MAX_INPUT	399	mm_FIRM	385
may	2	MM_HALT	386
mblen()	736	MM_HARD	385
mbrlen()	738	MM_INFO	386
mbrtowc()	740	MM_NOCON	386
mbsinit()	742	MM_NOMSG	386

MM_NOSEV.....	386	MST7MDT.....	1549
MM_NOTOK.....	386	msync().	815
MM_NRECOV.....	385	MS_.....	16, 18
MM_NULLMC.....	385	MS_ASYNC.....	776, 815
MM_OK.....	386	MS_INVALIDATE.....	815-816
MM_OPSYS.....	385	MS_SYNC.....	776, 815
MM_PRINT.....	385, 387	multicast.....	67
MM_RECOVER.....	385	munlock().	770, 818
MM_SOFT.....	385	munlockall().	772, 819
MM_UTIL.....	385	munmap().	820
MM_WARNING.....	386	mutex attributes.....	1092
modf().	781	mutex initialization attributes.....	1091
modff.....	781	mutex performance.....	1092
modfl.....	781	MUXID_ALL.....	621-622
MON.....	6	MUXID_R.....	18
MORECTL.....	525	MX.....	6
MOREDATA.....	525	M_.....	18
MPR.....	6	name information.....	528
mprotect().	783	name space.....	14
MQ.....	16	NAME_MAX.....	25, 97, 171, 185, 187, 190
mq.....	16		300, 319, 338, 391, 399, 430, 449
mq_close().	785		458, 674, 683, 730, 755, 758, 762
mq_getattr().	786		791, 802, 830, 840, 845, 967, 1179
mq_notify().	788		1183, 1190, 1215, 1223, 1255, 1263
mq_open().	790		1330, 1333, 1404, 1470, 1542, 1562
MQ_OPEN_MAX.....	1473		1570, 1572
MQ_PRIO_MAX.....	796-797, 1473	NaN.....	123, 407, 471
mq_receive().	793	nan().	822
mq_send().	796	nanf.....	822
mq_setattr().	798	nanl.....	822
mq_timedreceive.....	793	nanosleep().	823
mq_timedreceive().	800	NDEBUG.....	21, 128
mq_timedsend.....	796	nearbyint().	825
mq_timedsend().	801	nearbyintf.....	825
mq_unlink().	802	nearbyintl.....	825
rand48().	263, 804	network interfaces.....	59
MSG.....	6, 18	NEW_TIME.....	284-285
msgctl().	805	nextafter().	827
msgget().	807	nextafterf.....	827
msgrcv().	809	nextafterl.....	827
msgsnd().	812	nexttoward.....	827
MSGVERB.....	386-387	nexttowardf.....	827
msg_.....	16	nexttowardl.....	827
MSG_.....	18	nftw().	829
MSG_ANY.....	524	NGROUPS_MAX.....	513, 1473
MSG_BAND.....	524, 1154	nice().	832
MSG_EOR.....	1393, 1395	NLSPATH.....	171
MSG_HIPRI.....	524, 1154	NL_.....	18
MSG_NOERROR.....	809-810	NL_ARGMAX.....	404, 435, 468, 477
msg_perm.....	40	NL_CAT_LOCALE.....	171
msqid.....	40	nl_langinfo().	834

nohup utility	303	PS	7
non-local jumps	1372	RS	7
non-volatile storage	452	RTS	7
nrand48()	263, 836	SD	7
ntohl()	587, 837	SEM	8
ntohs	587, 837	SHM	8
NULL	215, 241, 248, 256, 261, 779, 1181	SIO	8
NUM_EMPL	585	SPI	8
NZERO	543, 832	SPN	8
n_	16	SS	8
OB	6	TCT	8
obsolescent	180	TEF	8
OF	7	THR	9
OH	7	TMO	9
OLD_TIME	284-285	TMR	9
open a file	842	TPI	9
open a named semaphore	1255	TPP	9
open a shared memory object	1330	TPS	9
open()	838	TRC	9
opendir()	845	TRI	10
openlog()	210, 848	TRL	10
OPEN_MAX	171, 265, 280, 332, 391	TSA	10
.....	430, 463, 507, 510, 521, 551, 593	TSF	10
.....	790, 830, 840, 845, 856, 883, 886	TSH	10
.....	1473, 1531	TSP	10
optarg	532, 849	TSS	10
opterr	532, 849	TYM	10
optind	532, 849	UP	11
option		XSR	11
ADV	3	optopt	532, 535, 849
AIO	3	optstring	535
BAR	3	orphaned process group	310
BE	4	O_	18
CD	4	O_ constants	
CPT	4	used in open()	838
CS	4	used in posix_openpt()	873
FD	4	O_ACCMODE	329
FR	4	O_APPEND	41, 116, 246, 343, 838, 1656
FSC	5	O_CREAT	230, 790-791, 802, 838-840
IP6	5		1248, 1254, 1329-1331
MC1	5	O_DSYNC	108, 838-839, 1173, 1657
MC2	5	O_EXCL	791, 838-839, 1254, 1329-1330
MC3	5	O_NDELAY	1661
MF	5	O_NOCTTY	839, 843, 873
ML	6	O_NONBLOCK	24, 205, 327, 360, 364
MLR	6		370, 416, 420, 443, 445, 525, 617
MON	6		619, 791, 793, 796, 798, 839-841
MPR	6		856, 859, 1155, 1172, 1657, 1660
MSG	6	O_RDONLY	251, 790, 838-840, 843, 1329, 1331
MX	6	O_RDWR	322, 705, 790, 838-842, 873, 1329, 1331
PIO	7	O_RSYNC	839, 1173

O_SYNC	108, 839, 1173, 1657
O_TRUNC	230, 839, 841, 843, 1330-1331
O_WRONLY	230, 322, 705, 790, 838-841, 843
PAGESIZE	43, 770, 816, 820, 986, 1475
PAGE_SIZE	1475
PATH	215
PATH environment variable	305
pathconf()	399, 850
PATH_MAX	25, 97, 171, 185, 187, 190
.....	251, 300, 319, 338, 391, 399, 430
.....	449, 458, 494, 575, 674, 683, 730
.....	755, 758, 762, 791, 802, 830, 840
.....	845, 967, 1183, 1190, 1215, 1223
.....	1255, 1263, 1330, 1333, 1404, 1470
.....	1477, 1542, 1562, 1570, 1572
pause()	851
pclose()	852
pd_	16
PENDIN	19
perror()	854
persistent connection (I_PLINK)	622
PF_	18
physical write	452
ph_	16
PIO	7
pipe	396, 843, 1660
pipe()	856
PIPE_BUF	399, 1657, 1660
PIPE_MAX	1662
plain characters	1424
POLL	18
poll()	858
POLLERR	858
POLLHUP	858
POLLIN	858
POLLNVAL	859
POLLOUT	858
POLLPRI	858
POLLRDBAND	858
POLLRDNORM	858
POLLWRBAND	858
POLLWRNORM	858
POLL_	18
popen()	862
portability	3
POSIX	268
POSIX.1 symbols	13
POSIX_	15, 17
posix_	15, 17
POSIX_ALLOC_SIZE_MIN	399
posix_fadvise()	864
POSIX_FADV_DONTNEED	864
POSIX_FADV_NOREUSE	864
POSIX_FADV_NORMAL	864
POSIX_FADV_RANDOM	864
POSIX_FADV_SEQUENTIAL	864
POSIX_FADV_WILLNEED	864
posix_fallocate()	866
posix_madvise()	868
POSIX_MADV_DONTNEED	868
POSIX_MADV_NORMAL	868
POSIX_MADV_RANDOM	868
POSIX_MADV_SEQUENTIAL	868
POSIX_MADV_WILLNEED	868
posix_memalign()	872
posix_mem_offset()	870
posix_openpt()	873
POSIX_REC_INCR_XFER_SIZE	399
POSIX_REC_MAX_XFER_SIZE	399
POSIX_REC_MIN_XFER_SIZE	399
POSIX_REC_XFER_ALIGN	399
posix_spawn()	875
posix_spawnattr_destroy()	890
posix_spawnattr_getflags()	892
posix_spawnattr_getpgroup()	894
posix_spawnattr_getschedparam()	896
posix_spawnattr_getschedpolicy()	898
posix_spawnattr_getsigdefault()	900
posix_spawnattr_getsigmask()	902
posix_spawnattr_init()	890, 904
posix_spawnattr_setflags()	892, 905
posix_spawnattr_setpgroup()	894, 906
posix_spawnattr_setschedparam()	896, 907
posix_spawnattr_setschedpolicy()	898, 908
posix_spawnattr_setsigdefault()	900, 909
posix_spawnattr_setsigmask()	902, 910
posix_spawnp()	875, 911
posix_spawn_file_actions_addclose()	883
posix_spawn_file_actions_adddup2()	886
posix_spawn_file_actions_addopen()	883, 888
posix_spawn_file_actions_destroy()	889
posix_spawn_file_actions_init	889
POSIX_SPAWN_RESETIDS	876, 892
POSIX_SPAWN_SETPGROUP	876, 892, 894
POSIX_SPAWN_SETSCHEDPARAM	892, 896
POSIX_SPAWN_SETSCHEDULER	876, 892
.....	896, 898
POSIX_SPAWN_SETSIGDEF	877, 892, 900
POSIX_SPAWN_SETSIGMASK	892, 902
POSIX_TRACE_ADD_EVENTSET	950
POSIX_TRACE_ALL_EVENTS	944
POSIX_TRACE_APPEND	917, 937

<code>posix_trace_attr_destroy()</code>	912
<code>posix_trace_attr_getclockres()</code>	914
<code>posix_trace_attr_getcreatetime</code>	914
<code>posix_trace_attr_getgenversion</code>	914
<code>posix_trace_attr_getinherited()</code>	916
<code>posix_trace_attr_getlogfullpolicy</code>	916
<code>posix_trace_attr_getlogsize()</code>	919
<code>posix_trace_attr_getmaxdatasize</code>	919
<code>posix_trace_attr_getmaxsystemeventsz</code>	919
<code>posix_trace_attr_getmaxusereventsz</code>	919
<code>posix_trace_attr_getname()</code>	914, 922
<code>posix_trace_attr_getstreamfullpolicy()</code>	916, 923
<code>posix_trace_attr_getstreamsize()</code>	919, 924
<code>posix_trace_attr_init()</code>	912, 925
<code>posix_trace_attr_setinherited()</code>	916, 926
<code>posix_trace_attr_setlogfullpolicy</code>	916, 926
<code>posix_trace_attr_setlogsize()</code>	919, 927
<code>posix_trace_attr_setmaxdatasize</code>	919, 927
<code>posix_trace_attr_setname()</code>	914, 928
<code>posix_trace_attr_setstreamfullpolicy()</code>	916, 929
<code>posix_trace_attr_setstreamsize()</code>	919, 930
<code>posix_trace_clear()</code>	931
<code>posix_trace_close()</code>	933
<code>POSIX_TRACE_CLOSE_FOR_CHILD</code>	916
<code>posix_trace_create()</code>	935
<code>posix_trace_create_withlog</code>	935
<code>POSIX_TRACE_ERROR</code> trace event	77
<code>posix_trace_event()</code>	939
<code>posix_trace_eventid_equal()</code>	941
<code>posix_trace_eventid_get_name</code>	941
<code>posix_trace_eventid_open()</code>	939, 943
<code>posix_trace_eventset_add()</code>	944
<code>posix_trace_eventset_del</code>	944
<code>posix_trace_eventset_empty</code>	944
<code>posix_trace_eventset_fill</code>	944
<code>posix_trace_eventset_ismember</code>	944
<code>posix_trace_eventtypelist_getnext_id()</code>	946
<code>posix_trace_eventtypelist_rewind</code>	946
<code>posix_trace_event_info</code> structure	74
<code>POSIX_TRACE_FILTER</code> trace event	77, 950
<code>POSIX_TRACE_FLUSH</code>	917
<code>posix_trace_flush()</code>	935, 947
<code>POSIX_TRACE_FLUSHING</code>	73
<code>POSIX_TRACE_FULL</code>	72-74
<code>posix_trace_getnext_event()</code>	953
<code>posix_trace_get_attr()</code>	948
<code>posix_trace_get_filter()</code>	950
<code>posix_trace_get_status()</code>	948, 952
<code>POSIX_TRACE_INHERITED</code>	916
<code>POSIX_TRACE_LOOP</code>	73, 916-917, 937
<code>POSIX_TRACE_NOT_FLUSHING</code>	73
<code>POSIX_TRACE_NOT_FULL</code>	72-74
<code>POSIX_TRACE_NOT_FULL</code>	931
<code>POSIX_TRACE_NOT_TRUNCATED</code>	75, 954
<code>POSIX_TRACE_NO_OVERRUN</code>	73-74, 948
<code>posix_trace_open()</code>	933, 956
<code>POSIX_TRACE_OVERFLOW</code> trace event	77
<code>POSIX_TRACE_OVERRUN</code>	73-74
<code>POSIX_TRACE_RESUME</code> trace event	77
<code>posix_trace_rewind</code>	933, 956
<code>POSIX_TRACE_RUNNING</code>	72, 959
<code>POSIX_TRACE_SET_EVENTSET</code>	950
<code>posix_trace_set_filter()</code>	950, 957
<code>posix_trace_shutdown()</code>	935, 958
<code>POSIX_TRACE_START</code> trace event	77, 959
<code>posix_trace_start()</code>	959
<code>posix_trace_status_info</code> structure	72
<code>posix_trace_stop</code>	959
<code>POSIX_TRACE_STOP</code> trace event	77, 959
<code>POSIX_TRACE_SUB_EVENTSET</code>	950
<code>POSIX_TRACE_SUSPENDED</code>	72-73, 959
<code>POSIX_TRACE_SYSTEM_EVENTS</code>	944
<code>posix_trace_timedgetnext_event()</code>	953, 961
<code>posix_trace_trid_eventid_open()</code>	941, 962
<code>POSIX_TRACE_TRUNCATED_READ</code>	75, 954
<code>POSIX_TRACE_TRUNCATED_RECORD</code>	75, 954
<code>posix_trace_trygetnext_event()</code>	953, 963
<code>POSIX_TRACE_UNTIL_FULL</code>	73, 916-917, 937
<code>POSIX_TRACE_USER_EVENT_MAX</code>	939
<code>POSIX_TRACE_WOPID_EVENTS</code>	944
<code>POSIX_TYPED_MEM_ALLOCATE</code>	774-775
.....	870, 964, 966
<code>POSIX_TYPED_MEM_ALLOCATE_CONTIG</code>	774-775, 870, 964, 966
<code>posix_typed_mem_get_info()</code>	964
<code>POSIX_TYPED_MEM_MAP_ALLOCATABLE</code>	820, 966
<code>posix_typed_mem_open()</code>	966
<code>pow()</code>	969
<code>powf</code>	969
<code>powl</code>	969
<code>pread()</code>	972 , 1172
predefined stream	
standard error	37
standard input	37
standard output	37
preempted thread	1038
<code>PRI</code>	18
<code>printf()</code>	404, 973
priority	38
<code>PRIOR</code>	18
<code>PRIOR_INHERIT</code>	1088

PRIO_PGRP	543	pthread_barrierattr_destroy()	1016
PRIO_PROCESS	543	pthread_barrierattr_getpshared()	1018
PRIO_USER	543	pthread_barrierattr_init()	1016, 1020
process		pthread_barrierattr_setpshared()	1018, 1021
concurrent execution	396	pthread_barrier_destroy()	1012
setting real and effective user IDs	1313	pthread_barrier_init	1012
single-threaded	396	PTHREAD_BARRIER_SERIAL_THREAD	1014
process creation	396	pthread_barrier_wait()	1014
process group		pthread_cancel()	1022
orphaned	310	PTHREAD_CANCELED	57, 1057
process group ID	539, 1305, 1317	PTHREAD_CANCEL_ASYNCHRONOUS	54
process ID	310		1132
process lifetime	670	PTHREAD_CANCEL_DEFERRED	54, 1036, 1132
process scheduling	44	PTHREAD_CANCEL_DISABLE	54, 1132
process shared memory	1092	PTHREAD_CANCEL_ENABLE	54, 1132
process synchronization	1092	pthread_cleanup_pop()	1024
process termination	309	pthread_cleanup_push	1024
PROT_	16, 18	pthread_condattr_destroy()	1041
PROT_EXEC	774, 783	pthread_condattr_getclock()	1043
PROT_NONE	44, 774, 783	pthread_condattr_getpshared()	1045
PROT_READ	774, 783	pthread_condattr_init()	1041, 1047
PROT_WRITE	774, 776, 778, 783	pthread_condattr_setclock()	1043, 1048
PS	7	pthread_condattr_setpshared()	1045, 1049
pselect()	974	pthread_cond_broadcast()	1029
pseudo-random sequence generation functions	1169	pthread_cond_destroy()	1032
PST8PDT	1549	pthread_cond_init	1032
ps_	16	PTHREAD_COND_INITIALIZER	1032
PTHREAD_	16	pthread_cond_signal()	1029, 1035
pthread_	16	pthread_cond_timedwait()	1036
pthread_atfork()	979	pthread_cond_wait	1036
pthread_attr_destroy()	981	pthread_create()	1050
pthread_attr_getdetachstate()	984	PTHREAD_CREATE_DETACHED	30, 984
pthread_attr_getguardsize()	986	PTHREAD_CREATE_JOINABLE	30, 984, 1067
pthread_attr_getinheritsched()	988	PTHREAD_DESTRUCTOR_ITERATIONS	1064
pthread_attr_getschedparam()	990		1069, 1475
pthread_attr_getschedpolicy()	992	pthread_detach()	1053
pthread_attr_getscope()	994	pthread_equal()	1055
pthread_attr_getstack()	996	pthread_exit()	1056
pthread_attr_getstackaddr()	998	PTHREAD_EXPLICIT_SCHED	988
pthread_attr_getstacksize()	1000	pthread_getconcurrency()	1058
pthread_attr_init()	981, 1002	pthread_getcpuclockid()	1060
pthread_attr_setdetachstate()	984, 1003	pthread_getschedparam()	1061
pthread_attr_setguardsize()	986, 1004	pthread_getspecific()	1064
pthread_attr_setinheritsched()	988, 1005	PTHREAD_INHERIT_SCHED	988
pthread_attr_setschedparam()	990, 1006	pthread_join()	1066
pthread_attr_setschedpolicy()	992, 1007	PTHREAD_KEYS_MAX	1069, 1475
pthread_attr_setscope()	994, 1008	pthread_key_create()	1069
pthread_attr_setstack()	996, 1009	pthread_key_delete()	1073
pthread_attr_setstackaddr()	998, 1010	pthread_kill()	1075
pthread_attr_setstacksize()	1000, 1011	pthread_mutexattr_destroy()	1091
		pthread_mutexattr_getprioceiling()	1096

<code>pthread_mutexattr_getprotocol()</code>	1098
<code>pthread_mutexattr_getpshared()</code>	1100
<code>pthread_mutexattr_gettype()</code>	1102
<code>pthread_mutexattr_init()</code>	1091, 1104
<code>pthread_mutexattr_setprioceiling()</code>	1096, 1105
<code>pthread_mutexattr_setprotocol()</code>	1098, 1106
<code>pthread_mutexattr_setpshared()</code>	1100, 1107
<code>pthread_mutexattr_settype()</code>	1102, 1108
<code>PTHREAD_MUTEX_DEFAULT</code>	1084, 1102
<code>pthread_mutex_destroy()</code>	1076
<code>PTHREAD_MUTEX_ERRORCHECK</code>	1084, 1102
<code>pthread_mutex_getprioceiling()</code>	1081
<code>pthread_mutex_init()</code>	1076, 1083
<code>PTHREAD_MUTEX_INITIALIZER</code>	1076, 1083
<code>pthread_mutex_lock()</code>	1084
<code>PTHREAD_MUTEX_NORMAL</code>	1084, 1102
<code>PTHREAD_MUTEX_RECURSIVE</code>	1084
.....	1102-1103
<code>pthread_mutex_setprioceiling()</code>	1081, 1087
<code>pthread_mutex_timedlock()</code>	1088
<code>pthread_mutex_trylock()</code>	1084, 1090
<code>pthread_mutex_unlock</code>	1084, 1090
<code>pthread_once()</code>	1109
<code>PTHREAD_ONCE_INIT</code>	1109
<code>PTHREAD_PRIO_INHERIT</code>	1098
<code>PTHREAD_PRIO_NONE</code>	1098
<code>PTHREAD_PRIO_PROTECT</code>	1085, 1098
<code>PTHREAD_PROCESS_PRIVATE</code>	1018, 1045
.....	1092, 1100, 1127, 1141
<code>PTHREAD_PROCESS_SHARED</code>	1018, 1045
.....	1092, 1100, 1127, 1141
<code>pthread_rwlockattr_destroy()</code>	1125
<code>pthread_rwlockattr_getpshared()</code>	1127
<code>pthread_rwlockattr_init()</code>	1125, 1129
<code>pthread_rwlockattr_setpshared()</code>	1127, 1130
<code>pthread_rwlock_destroy()</code>	1111
<code>pthread_rwlock_rdlock()</code>	1111, 1113
<code>pthread_rwlock_timedrdlock()</code>	1115
<code>pthread_rwlock_timedwrlock()</code>	1117
<code>pthread_rwlock_tryrdlock()</code>	1113, 1119
<code>pthread_rwlock_trywrlock()</code>	1120
<code>pthread_rwlock_unlock()</code>	1122
<code>pthread_rwlock_wrlock()</code>	1120, 1124
<code>PTHREAD_SCOPE_PROCESS</code>	52, 994
<code>PTHREAD_SCOPE_SYSTEM</code>	52, 994
<code>pthread_self()</code>	1131
<code>pthread_setcancelstate()</code>	1132
<code>pthread_setcanceltype</code>	1132
<code>pthread_setconcurrency()</code>	1058, 1134
<code>pthread_setschedparam()</code>	1061, 1135
<code>pthread_setschedprio()</code>	1136
<code>pthread_setspecific()</code>	1064, 1138
<code>pthread_sigmask()</code>	1139
<code>pthread_spin_destroy()</code>	1141
<code>pthread_spin_init</code>	1141
<code>pthread_spin_lock()</code>	1143
<code>pthread_spin_trylock</code>	1143
<code>pthread_spin_unlock()</code>	1145
<code>PTHREAD_STACK_MIN</code>	996, 998, 1000, 1475
<code>pthread_testcancel()</code>	1132, 1146
<code>PTHREAD_THREADS_MAX</code>	1050, 1475
<code>ptsname()</code>	1147
<code>putc()</code>	1148
<code>putchar()</code>	1150
<code>putchar_unlocked()</code>	487, 1151
<code>putc_unlocked()</code>	487, 1149
<code>putenv()</code>	1152
<code>putmsg()</code>	1154
<code>putpmsg</code>	1154
<code>puts()</code>	1158
<code>pututxline()</code>	284, 1160
<code>putwc()</code>	1161
<code>putwchar()</code>	1162
<code>pwrite()</code>	1163 , 1656
<code>pw_</code>	16
<code>p_</code>	16
<code>P_</code>	17
<code>P_ALL</code>	1595
<code>P_PGID</code>	1595
<code>P_PID</code>	1595
<code>qsort()</code>	1164
queue a signal to a process	1370
<code>raise()</code>	1166
<code>rand()</code>	1168
<code>random()</code>	608, 1171
<code>RAND_MAX</code>	1168
<code>rand_r</code>	1168
read from a file	1175
<code>read()</code>	1172
<code>readdir()</code>	1179
<code>readdir_r</code>	1179
<code>readlink()</code>	1183
<code>readv()</code>	1186
real user ID	98, 669
<code>realloc()</code>	1188
<code>realpath()</code>	1190
<code>REALTIME</code>	105, 107-108, 110, 113-114
.....	116, 197, 203, 337, 686, 770, 772
.....	785-786, 788, 790, 793, 796, 798
.....	802, 823, 1233-1237, 1240, 1248-1250
.....	1252, 1254, 1257, 1261, 1263, 1265
.....	1329, 1333, 1369, 1376, 1382, 1521

.....	1524-1525	rindex().....	1220
REALTIME THREADS	988, 992, 994	rint().....	1221
.....	1005, 1007-1008, 1061, 1081, 1087	RLIMIT_.....	18
.....	1096, 1098, 1105-1106, 1135-1136	RLIMIT_AS	555
recv().....	1192	RLIMIT_CORE.....	554
recvfrom().....	1194	RLIMIT_CPU.....	554
recvmsg().....	1197	RLIMIT_DATA	554
regcomp().....	1200	RLIMIT_FSIZE	554
regerror.....	1200	RLIMIT_NOFILE.....	554, 556
regexec.....	1200	RLIMIT_STACK.....	554
regfree.....	1200	rlim_.....	16
register fork handlers.....	979	RLIM_.....	18
REG_.....	18	RLIM_INFINITY.....	554-555
REG_ constants		RLIM_SAVED_CUR.....	555
error return values of regcomp	1202	RLIM_SAVED_MAX.....	555
used in regcomp	1200	rmdir().....	1223
REG_BADBR	1202	RMSGD.....	616
REG_BADPAT	1202	RMSGN.....	616
REG_BADRPT	1202	RNORM.....	616
REG_EBRACE	1202	round robin	46
REG_EBRACK	1202	round().....	1226
REG_ECOLLATE.....	1202	roundf.....	1226
REG_ECTYPE.....	1202	roundl.....	1226
REG_EESCAPE	1202	routing.....	59
REG_EPAREN.....	1202	RPROTDAT.....	616
REG_ERANGE.....	1202	RPROTDIS	616
REG_ESPACE.....	1202	RPROTNORM.....	616
REG_ESUBREG.....	1202	RS	7
REG_EXTENDED.....	1200	RS_HIPRI.....	524, 615, 1154
REG_ICASE	1200	RTLD_.....	18
REG_NEWLINE	1200	RTLD_DEFAULT	261
REG_NOMATCH	1202	RTLD_GLOBAL	254, 258-259, 262
REG_NOSUB.....	1200	RTLD_LAZY	258, 261
REG_NOTBOL.....	1201	RTLD_LOCAL	259
REG_NOTEOL.....	1201	RTLD_NEXT	261-262
remainder().....	1207	RTLD_NOW	258-259
remainderf.....	1207	RTS.....	7
remainderl.....	1207	RTSIG_MAX.....	1475
remove a directory	1224	RUSAGE_.....	18
remove directory entries.....	1564	RUSAGE_CHILDREN.....	557
remove().....	1209	RUSAGE_SELF	557
remque().....	610, 1211	ru_.....	16
remquo().....	1212	s6_.....	16
remquof.....	1212	sa_.....	16
remquol.....	1212	SA_.....	18
rename a file	1216	SA_NOCLDSTOP.....	31, 1345, 1348
rename().....	1214	SA_NOCLDWAIT	307-308, 557, 1346, 1589
rewind().....	1218	SA_NODEFER	1346
rewinddir().....	1219	SA_ONSTACK.....	297, 1345
re_.....	16	SA_RESETHAND	148, 1345-1346
RE_DUP_MAX.....	1475	SA_RESTART	148, 977, 1345, 1359

SA_SIGINFO	1344-1345, 1348, 1369
scalb()	1228
scalbln()	1230
scalblnf	1230
scalblnl	1230
scalbn	1230
scalbnf	1230
scalbnl	1230
scanf()	435, 1232
schedule alarm	119
scheduling documentation	54
scheduling policy	
round robin	46
SCHED_	16
sched_	16
SCHED_FIFO	42, 45, 53, 298, 394
.....	543, 832, 990, 992, 1061, 1096
.....	1113, 1238, 1257
sched_getparam()	1234
sched_getscheduler()	1235
sched_get_priority_max()	1233
sched_get_priority_min	1233
SCHED_OTHER	45, 48, 543, 992, 1061, 1238
SCHED_RR	42, 45-46, 53, 298, 394
.....	543, 832, 990, 992, 1061, 1113, 1238, 1257
sched_rr_get_interval()	1236
sched_setparam()	1237
sched_setscheduler()	1240
SCHED_SPORADIC	42, 45-46, 1113, 1238, 1257
sched_yield()	1243
SCM_	18
SCN	18
SD	7
security considerations	191, 309, 669, 1305, 1406
seed48()	263, 1244
seekdir()	1245
SEEK_CUR	330, 442, 728
SEEK_END	330, 442, 728
SEEK_GET	1218
SEEK_SET	42, 110, 116, 330, 442, 728
SEGV_	18
select()	974, 1247
SEM	8
semctl()	1266
semget()	1269
semid	40
semop()	1272
SEM_	16, 18
sem_	16
sem_close()	1248
sem_destroy()	1249
SEM_FAILED	1255
sem_getvalue()	1250
sem_init()	1252
SEM_NSEMS_MAX	1252, 1475
sem_open()	1254
sem_perm	40
sem_post()	1257
sem_timedwait()	1259
sem_trywait()	1261
SEM_UNDO	1272
sem_unlink()	1263
SEM_VALUE_MAX	1254, 1475
sem_wait()	1261, 1265
send()	1277
sendmsg()	1279
sendto()	1282
service name	426
session	310, 669, 1305, 1317
set cancelability state	1132
set file creation mask	1555
set process group ID for job control	1305
set-group-ID	189, 304, 309, 334
set-user-ID	304, 309, 494, 669
SETALL	1266, 1269
setbuf()	1285
setcontext()	491, 1286
setgid()	1287
setenv()	1288
seteuid()	1290
setgid()	1291
setgrent()	272, 1293
sethostent()	274, 1294
setitimer()	519, 1295
setjmp()	1296
setkey()	1298
setlocale()	1299
setlogmask()	210, 1303
setnetent()	276, 1304
setpgid()	1305
setpgrp()	1307
setpriority()	543, 1308
setprotoent()	278, 1309
setpwent()	280, 1310
setregid()	1311
setreuid()	1313
setrlimit()	554, 1315
setservent()	282, 1316
setsid()	1317
setsockopt()	1319
setstate()	608, 1322
setuid()	1323

- setutxent() 284, **1326**
- SETVAL 1266, 1269
- setvbuf()**1327**
- shall 2
- shell 302, 310, 522, 539, 669, 1306, 1592
 - job**669**
 - login 522
- shell scripts
 - exec 303
- shell, login 302
- SHM**8**, 18
- shmat()**1335**
- shmctl()**1337**
- shmdt()**1339**
- shmget()**1340**
- shmid 40
- SHMLBA 1335
- shm_ 16
- SHM_ 18
- shm_open()**1329**
- shm_perm 40
- SHM_RDONLY 1335
- SHM_RND 1335
- shm_unlink()**1333**
- should 2
- shutdown()**1342**
- SHUT_ 18
- SIGABRT 92
- sigaction()**1344**
- sigaddset()**1350**
- SIGALRM 119, 519, 1388, 1551, 1568
- sigaltstack()**1351**
- SIGBUS 44, 776, 779, 1139
- SIGCANCEL 1022
- SIGCHLD 211, 307-308, 557, 582
 - 1345, 1348, 1357, 1481, 1589, 1595
- SIGCLD 1348
- SIGCONT 34, 308, 310, 668-669
- sigdelset()**1353**
- sigemptyset()**1354**
- SIGEV_ 16
- sigev_ 16
- SIGEV_NONE 29, 42
- SIGEV_SIGNAL 29, 1521
- SIGEV_THREAD 29-30
- sigfillset()**1356**
- SIGFPE 1139, 1363
- sighold()**1357**
- SIGHUP 205, 308, 310
- sigignore 1357
- SIGILL 1139, 1363
- SIGINT 396, 1481
- siginterrupt()**1359**
- sigismember()**1361**
- SIGKILL 669, 1344, 1348, 1357
- siglongjmp()**1362**
- signal generation and delivery 28
 - realtime 29
- signal handler 1363
- signal()**1363**
- signaling a condition 1030
- signals 28
- signbit()**1365**
- sigpause() 1357, **1366**
- sigpending()**1367**
- SIGPIPE 327, 360, 416, 420, 443, 446, 1155, 1659
- SIGPOLL 205, 614-615
- sigprocmask() 1139, **1368**
- SIGPROF 519
- sigqueue()**1369**
- SIGQUEUE_MAX 1369, 1475
- SIGQUIT 1481
- sigrelse() 1357, **1371**
- SIGRTMAX 29-30, 1369, 1376, 1380
- SIGRTMIN 29-30, 1369, 1376, 1380
- SIGSEGV 44, 555, 820, 986, 1139, 1363
- sigset 1357, 1371
- sigsetjmp()**1372**
- SIGSTKSZ 1351
- SIGSTOP 29, 1344, 1348, 1357
- sigsuspend()**1374**
- sigtimedwait()**1376**
- SIGTSTP 29
- SIGTTIN 29, 364, 370, 1174
- SIGTTOU 29, 327, 360, 416, 420
 - 443, 445, 1490, 1492, 1494, 1501
 - 1504, 1659
- SIGURG 615
- SIGVTALRM 519
- sigwait()**1380**
- sigwaitinfo() 1376, **1382**
- SIGXCPU 554
- SIGXFSZ 554, 1542
- SIG_ 16, 18
- SIG_BLOCK 1139
- SIG_DFL 30, 297, 555, 1344, 1346, 1363
- SIG_ERR 148, 1363
- SIG_HOLD 1357
- SIG_IGN 30-31, 297, 303, 307-308, 557
 - 1344, 1363, 1589
- SIG_SETMASK 1139
- SIG_UNBLOCK 1139

sin()	1383	SPI	8
sin6_	16	SPN	8
sinf	1383	sporadic server policy	
sinh()	1385	execution capacity	46
sinhf	1385	replenishment period	46
sinhl	1385	sprintf()	404, 1397
sinl()	1383, 1387	spurious wakeup	1030
sin_	16	sqrt()	1398
SIO	8	sqrtf	1398
SIOCATMARK	1391	sqrtr	1398
sival_	16	srand()	1168, 1400
SI_	16, 18	srand48()	263, 1401
si_	16-17	random()	608, 1402
SI_ASYNCIO	32	SS	8
SI_MSGQ	32	sscanf()	435, 1403
SI_QUEUE	32	SSIZE_MAX	793, 809, 1172, 1183, 1425, 1656
SI_TIMER	32	ss_	16-17
SI_USER	32	SS_	18
sleep()	1388	SS_DISABLE	1351-1352
sl_	16	SS_ONSTACK	1351
SND	18	stack size	981
SNDZERO	618	stat()	1404
snprintf()	404, 1390	statvfs()	449, 1408
SO	18	stderr	1409
socketatmark()	1391	STDERR_FILENO	1409
socket I/O mode	60	stdin	1409
socket out-of-band data	61	STDIN_FILENO	862, 1409
socket owner	60	stdio locking functions	375
socket queue limits	60	stdio with explicit client locking	487
socket receive queue	61	stdout	1409
socket types	59	STDOUT_FILENO	862, 1409
socket()	1393	STR	18
socketpair()	1395	strcasecmp()	1411
sockets	58	strcat()	1412
address families	58	strchr()	1413
addressing	59	strcmp()	1414
asynchronous errors	62	strcoll()	1416
connection indication queue	62	strcpy()	1418
Internet Protocols	66	strcspn()	1420
IPv4	67	strdup()	1421
IPv6	67	STREAM	617, 619, 1154, 1173, 1658
local UNIX connections	66	stream	
options	63	byte-oriented	36
pending error	60	wide-oriented	36
protocols	59	STREAM head / tail	38
signals	62	stream-full-policy attribute	72-73, 76, 917
SOCK_	18	stream-min-size attribute	76, 920
SOCK_DGRAM	66, 1393, 1395	STREAMS	22, 205, 319, 338, 524, 613
SOCK_RAW	66		628, 840-841, 858, 974
SOCK_SEQPACKET	66, 1393, 1395	access	39
SOCK_STREAM	66, 1393, 1395	multiplexed	621

overview.....	38	SYMLINK_MAX	399, 1470
streams.....	34	SYMLOOP_MAX	147, 220, 320, 338, 391
interaction with file descriptors	35		431, 450, 458, 464, 674, 762, 830
stream orientation	36		1190, 1281, 1284, 1475, 1543, 1572
STREAM_MAX.....	342, 391, 862, 1475	sync()	1472
strerror().....	1422	synchronously accept a signal.....	1377
strerror_r	1422	sysconf().....	1473
strfmon().....	1424	syslog().....	210, 1480
strftime().....	1428	system crash	452
strlen().....	1434	System III	191, 1557
strncasecmp().....	1411, 1436	system interfaces	83
strncat().....	1437	system name	1557
strncmp().....	1438	system trace event type definitions	76
strncpy().....	1439	System V.....	119, 191, 305, 310, 332-333
strpbrk().....	1440		401, 539, 669, 756, 1224, 1317
strptime().....	1441		1348, 1372, 1496, 1557
strrchr().....	1445	system().....	1481
strspn().....	1446	s_	16
strstr().....	1447	S_	18
strtod().....	1448	S_BANDURG	615
strtof	1448	S_ERROR.....	614
strtoimax().....	1452	S_HANGUP.....	615
strtok().....	1453	S_HIPRI	614
strtok_r.....	1453	S_IFBLK	761
strtol().....	1456	S_IFCHR.....	761
strtold().....	1448, 1458	S_IFDIR.....	761
strtoll().....	1456, 1459	S_IFIFO	761
strtoul().....	1460	S_IFREG.....	761
strtoull.....	1460	S_INPUT.....	614
strtoumax()	1452, 1463	S_IRGRP.....	323, 447, 761
strxfrm().....	1464	S_IROTH.....	323, 447, 761
str_	16	S_IRUSR.....	323, 447, 761
st_	16	S_IRWXG	761
ST_	18	S_IRWXO	761
ST_NOSUID	298, 449	S_IRWXU	761
ST_RDONLY	449	S_ISGID.....	187-189, 761, 1542, 1657
sun_	17	S_ISUID.....	187-188, 761, 1542, 1657
superuser.....	98, 191, 685, 1564	S_ISVTX	187, 761, 1215, 1224, 1562
supplementary groups.....	191, 512	S_IWGRP	323, 447, 761
SVID	1372	S_IWOTH	323, 447, 761
SVR4.....	778, 823	S_IWUSR	323, 447, 761
sv_	16	S_IXGRP	761
SV_	18	S_IXOTH	761
swab().....	1466	S_IXUSR	761
swapcontext().....	732, 1467	S_MSG.....	614
swprintf()	468, 1468	S_OUTPUT	614
swscanf()	477, 1469	S_RDBAND.....	614-615
SWTCH	19	S_RDNORM	614
symbols		S_WRBAND	614
POSIX.1.....	13	S_WRNORM	614
symlink().....	1470	TABSIZE	150, 726

tan()	1485	thread-specific data management	1065
tanf	1485	threads	50
tanh()	1487	regular file operations	58
tanhf	1487	time()	1518
tanhf	1487	timer ID	1523
tanl()	1485, 1489	TIMER_	17-18
tcdrain()	1490	timer_	17
tcflow()	1492	TIMER_ABSTIME	49, 200, 1525
tcflush()	1494	timer_create()	1521
tcgetattr()	1496	timer_delete()	1524
tcgetpgrp()	1498	timer_getoverrun()	1525
tcgetsid()	1500	timer_gettime	1525
TCIFLUSH	1494	TIMER_MAX	1475
TCIOFF	1492	timer_settime	1525
TCIOFLUSH	1494	times()	1528
TCION	1492	timezone()	1530
TCOFLUSH	1494	TMO	9
TCOOFF	1492	tmpfile()	1531
TCOON	1492	tmpnam()	1533
TCP_	18	TMP_MAX	1513, 1532-1533
TCSADRAIN	1503	TMR	9
TCSAFLUSH	1503	tms_	16
TCSANOW	1503	tm_	17
tcsendbreak()	1501	toascii()	1535
tcsetattr()	1503	tolower()	1536
tcsetpgrp()	1506	TOSTOP	327, 360, 416, 420, 443, 445, 1658
TCT	8	toupper()	1537
tdelete()	1508	towctrans()	1538
TEF	8	towlower()	1539
telldir()	1512	towupper()	1540
tempnam()	1513	TPI	9
terminal access control	1496, 1504	TPP	9
terminal device name	1546	TPS	9
terminate a process	309	trace event, POSIX_TRACE_ERROR	77
terminology	1	trace event, POSIX_TRACE_FILTER	77, 950
termios structure	1496	trace event, POSIX_TRACE_OVERFLOW	77
tfind()	1508, 1515	trace event, POSIX_TRACE_RESUME	77
tgamma()	1516	trace event, POSIX_TRACE_START	77, 959
tgammaf	1516	trace event, POSIX_TRACE_STOP	77, 959
tgammaf	1516	trace functions	79
THR	9	trace-name attribute	76, 914
thread creation	1051	TRACE_EVENT_NAME_MAX	939, 941
thread creation attributes	981	TRACE_SYS_MAX	936
thread ID	1055	TRACE_USER_EVENT_MAX	939, 941
thread IDs	51	TRACING	912, 914, 916, 919, 922-931
thread mutexes	51		933, 935, 939, 941, 943-944, 946-948
thread scheduling	52		950, 952-953, 956-959, 961-963
thread termination	1056	TRAP_	18
thread-safety	50, 375	TRC	9
thread-specific data key creation	1071	TRI	10
thread-specific data key deletion	1073	TRL	10

trunc()	1541	UTC	1549
truncate()	1542	utime()	1570
truncation-status attribute	939	utimes()	1572
truncf	1541	utim_	17
truncf()	1544	uts_	17
truncl	1541, 1544	ut_	17
TSA	10	va_arg()	1574
tsearch()	1508, 1545	va_copy	1574
TSF	10	va_end	1574
TSH	10	va_start	1574
TSP	10	VDISCARD	19
TSS	10	VDSUSP	19
ttyname()	1546	Version 7	119, 191, 669, 1557
ttyname_r	1546	vfork()	1575
TTY_NAME_MAX	1475, 1546	vfprintf()	1577
tv_	16-17	vfprintf()	1578
twalk()	1508, 1548	vfwprintf()	1579
TYM	10	vfwscanf()	1580
tzname	1549	VISIT	1508, 1548
TZNAME_MAX	1475	VLNEXT	19
tzset()	1549	vprintf()	1577, 1581
t_uscalar_t	616	VREPRINT	19
ualarm()	1551	vscanf()	1578, 1582
uc_	16-17	vsprintf()	1577, 1583
UINT_MAX	119, 1389	vsprintf	1577, 1583
UIO_MAXIOV	17	vsscanf()	1578, 1584
ulimit()	1553	VSTATUS	19
ULLONG_MAX	1461	vswprintf()	1579, 1585
ULONG_MAX	1461, 1632	vswscanf()	1580, 1586
UL_	17	VWERASE	19
UL_GETFSIZE	1553	vwprintf()	1579, 1587
UL_SETFSIZE	1553	vwscanf()	1580, 1588
umask()	1555	wait for process termination	1592
uname()	1557	wait for thread termination	1067
undefined	2	wait()	1589
underlying function	36	waitid()	1595
ungetc()	1559	waiting on a condition	1037
ungetwc()	1560	waitpid()	1589, 1597
unicast	67	WARNING	386
unlink()	1562	warning	
unlockpt()	1566	OB	6
unsetenv()	1567	OF	7
unspecified	2	WCONTINUED	1589, 1595
UP	11	wcrtomb()	1598
US-ASCII	627	wcscat()	1600
user ID		wcschr()	1601
real and effective	1313	wcscmp()	1602
setting real and effective	1313	wscoll()	1603
user trace event type definitions	79	wcscpy()	1604
USER_PROCESS	284-285	wcscspn()	1605
usleep()	1568	wcsftime()	1606

wcslen()	1608	WRDE_CMDSUB	1653
wcsncat()	1609	WRDE_DOOFFS	1652
wcsncmp()	1610	WRDE_NOCMD	1652
wcsncpy()	1611	WRDE_NOSPACE	1653
wcspbrk()	1612	WRDE_REUSE	1652
wcsrchr()	1613	WRDE_SHOWERR	1652
wcsrtombs()	1614	WRDE_SYNTAX	1653
wcsspn()	1616	WRDE_UNDEF	1652
wcsstr()	1617	write to a file	1660
wcstod()	1618	write()	1656
wcstof	1618	writew()	1664
wcstoimax()	1621	wscanf()	477, 1666
wcstok()	1622	WSTOPPED	1595
wcstol()	1624	WSTOPSIG	1590
wcstold()	1618, 1627	WTERMSIG	1590
wcstoll()	1624, 1628	WUNTRACED	1589, 1592
wcstombs()	1629	XSI	11
wcstoul()	1631	XSI interprocess communication	39
wcstoumax()	1621, 1634	XSR	11
wcswcs()	1635	X_OK	98
wcswidth()	1636	y0()	1667
wcsxfrm()	1637	y1	1667
wctob()	1639	yn	1667
wctomb()	1640	zombie process	307
wctrans()	1642		
wctype()	1643		
wcwidth()	1645		
WEOF	82, 650-651, 653-654, 656-663		
	1539-1540, 1560		
WEXITED	1595		
WEXITSTATUS	1590		
we_	17		
wide-oriented stream	36		
WIFCONTINUED	1590		
WIFEXITED	1590		
WIFSIGNALED	1590		
WIFSTOPPED	1590, 1593		
wmemchr()	1646		
wmemcmp()	1647		
wmemcpy()	1648		
wmemmove()	1649		
wmemset()	1650		
WNOHANG	1348, 1589, 1595		
WNOWAIT	1595		
wordexp()	1651		
wordfree	1651		
wprintf()	468, 1655		
WRDE_	18		
WRDE_APPEND	1652		
WRDE_BADCHAR	1653		
WRDE_BADVAL	1653		

IEEE Std 1003.1™, 2003 Edition

The Open Group Technical Standard
Base Specifications, Issue 6

Includes IEEE Std 1003.1™-2001 and IEEE Std 1003.1™-2001/Cor 1-2002

Information Technology — Portable Operating System Interface (POSIX®)

Shell and Utilities

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

and

The Open Group



THE *Open* GROUP

[This page intentionally left blank]

Abstract

This standard is simultaneously ISO/IEC 9945:2002, IEEE Std 1003.1-2001, and forms the core of the Single UNIX Specification, Version 3.

The IEEE Std 1003.1, 2003 Edition includes IEEE Std 1003.1-2001/Cor 1-2002 incorporated into IEEE Std 1003.1-2001 (base document). The Corrigendum addresses problems discovered since the approval of IEEE Std 1003.1-2001. These changes are mainly due to resolving integration issues raised by the merger of the base documents that were incorporated into IEEE Std 1003.1-2001, which is the single common revision to IEEE Std 1003.1[™]-1996, IEEE Std 1003.2[™]-1992, ISO/IEC 9945-1:1996, ISO/IEC 9945-2:1993, and the Base Specifications of The Open Group Single UNIX[®] Specification, Version 2.

This standard defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. This standard is intended to be used by both applications developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of this standard:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

This standard describes the external characteristics and facilities that are of importance to applications developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, input/output (I/O), job control, network, portable operating system interface (POSIX[®]), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

Copyright © 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. All rights reserved. This printing is by the International Organization for Standardization with special permission of the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. Published in Switzerland.

Shell and Utilities, Issue 6

Published 31 March 2003 by the Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, U.S.A.
ISBN: 0-7381-3437-6 PDF 0-7381-3564-X / SS95078 CD-ROM 0-7381-3563-1 / SE95078
Printed in the United States of America by the IEEE.

Published 31 March 2003 by The Open Group
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, U.K.
Document Number: C033
ISBN: 1-931624-25-9
Printed in the U.K. by The Open Group.

All rights reserved. No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission from both the IEEE and The Open Group.

Portions of this standard are derived with permission from copyrighted material owned by Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc.

Permissions

Authorization to photocopy portions of this standard for internal or personal use is granted provided that the appropriate fee is paid to the Copyright Clearance Center or the equivalent body outside of the U.S. Permission to make multiple copies for educational purposes in the U.S. requires agreement and a license fee to be paid to the Copyright Clearance Center.

Beyond these provisions, permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests to the copyright holders should be sent by email to austin-group-permissions@opengroup.org.

Feedback

This standard has been prepared by the Austin Group. Feedback relating to the material contained in this standard may be submitted using the Austin Group web site at <http://www.opengroup.org/austin/defectform.html>.

IEEE

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property, or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “AS IS”.

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with the IEEE.¹ Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, U.S.A.

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE Standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

A patent holder has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and non-discriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Authorization to photocopy portions of any individual standard for internal or personal use is granted in the U.S. by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to the Copyright Clearance Center.² Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center. To arrange for payment of the licensing fee, please contact:

Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923, U.S.A., Tel.: +1 978 750 8400

Amendments, corrigenda, and interpretations for this standard, or information about the IEEE standards development process, may be found at <http://standards.ieee.org>.

Full catalog and ordering information on all IEEE publications is available from the IEEE Online Catalog & Store at <http://shop.ieee.org/store>.

1. For this standard, please send comments via the Austin Group as requested on page iii.

2. Please refer to the special provisions for this standard on page iii concerning permissions from both copyright holders and arrangements to cover photocopying and reproduction across the world, as well as by commercial organizations wishing to license the material for use in product documentation.

The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The Open Group's mission is to offer all organizations concerned with open information infrastructures a forum to share knowledge, integrate open initiatives, and certify approved products and processes in a manner in which they continue to trust our impartiality.

In the global eCommerce world of today, no single economic entity can achieve independence while still ensuring interoperability. The assurance that products will interoperate with each other across differing systems and platforms is essential to the success of eCommerce and business workflow. The Open Group, with its proven testing and certification program, is the international guarantor of interoperability in the new century.

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group's members include some of the largest and most influential organizations in the world. The flexible structure of The Open Groups membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information is available on The Open Group web site at <http://www.opengroup.org>.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes the *Westwood* family of tests for this standard and the associated certification program for Version 3 of the Single UNIX Specification, as well tests for CDE, CORBA, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at <http://www.opengroup.org/testing>.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at <http://www.opengroup.org/pubs>.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at <http://www.opengroup.org/corrigenda>.

Full catalog and ordering information on all Open Group publications is available at <http://www.opengroup.org/pubs>.

Contents

Chapter 1	Introduction.....	1
1.1	Scope.....	1
1.2	Conformance	1
1.3	Normative References	1
1.4	Change History	1
1.5	Terminology	1
1.6	Definitions	3
1.7	Relationship to Other Documents.....	3
1.7.1	System Interfaces	3
1.7.1.1	Process Attributes	3
1.7.1.2	Concurrent Execution of Processes.....	3
1.7.1.3	File Access Permissions	4
1.7.1.4	File Read, Write, and Creation	4
1.7.1.5	File Removal	6
1.7.1.6	File Time Values	6
1.7.1.7	File Contents	6
1.7.1.8	Pathname Resolution	7
1.7.1.9	Changing the Current Working Directory.....	7
1.7.1.10	Establish the Locale	7
1.7.1.11	Actions Equivalent to Functions.....	7
1.7.2	Concepts Derived from the ISO C Standard.....	7
1.7.2.1	Arithmetic Precision and Operations	7
1.7.2.2	Mathematical Functions	9
1.8	Portability	9
1.8.1	Codes	9
1.9	Utility Limits.....	17
1.10	Grammar Conventions	19
1.11	Utility Description Defaults.....	20
1.12	Considerations for Utilities in Support of Files of Arbitrary Size	27
1.13	Built-In Utilities	28
Chapter 2	Shell Command Language	29
2.1	Shell Introduction	29
2.2	Quoting	30
2.2.1	Escape Character (Backslash).....	30
2.2.2	Single-Quotes.....	30
2.2.3	Double-Quotes	30
2.3	Token Recognition	31
2.3.1	Alias Substitution	32
2.4	Reserved Words	33
2.5	Parameters and Variables.....	33
2.5.1	Positional Parameters.....	33

2.5.2	Special Parameters.....	34
2.5.3	Shell Variables.....	34
2.6	Word Expansions.....	36
2.6.1	Tilde Expansion.....	37
2.6.2	Parameter Expansion	37
2.6.3	Command Substitution	40
2.6.4	Arithmetic Expansion	41
2.6.5	Field Splitting.....	42
2.6.6	Pathname Expansion.....	42
2.6.7	Quote Removal.....	42
2.7	Redirection	43
2.7.1	Redirecting Input	44
2.7.2	Redirecting Output.....	44
2.7.3	Appending Redirected Output	44
2.7.4	Here-Document.....	44
2.7.5	Duplicating an Input File Descriptor.....	45
2.7.6	Duplicating an Output File Descriptor	45
2.7.7	Open File Descriptors for Reading and Writing.....	46
2.8	Exit Status and Errors	46
2.8.1	Consequences of Shell Errors	46
2.8.2	Exit Status for Commands	46
2.9	Shell Commands	47
2.9.1	Simple Commands	47
2.9.1.1	Command Search and Execution	48
2.9.2	Pipelines.....	49
2.9.3	Lists.....	50
2.9.3.1	Asynchronous Lists	50
2.9.3.2	Sequential Lists.....	51
2.9.3.3	AND Lists	51
2.9.3.4	OR Lists.....	51
2.9.4	Compound Commands.....	52
2.9.4.1	Grouping Commands	52
2.9.4.2	The for Loop.....	52
2.9.4.3	Case Conditional Construct	53
2.9.4.4	The if Conditional Construct.....	53
2.9.4.5	The while Loop.....	54
2.9.4.6	The until Loop	54
2.9.5	Function Definition Command	54
2.10	Shell Grammar.....	55
2.10.1	Shell Grammar Lexical Conventions.....	55
2.10.2	Shell Grammar Rules	56
2.11	Signals and Error Handling	61
2.12	Shell Execution Environment.....	61
2.13	Pattern Matching Notation	62
2.13.1	Patterns Matching a Single Character	62
2.13.2	Patterns Matching Multiple Characters	63
2.13.3	Patterns Used for Filename Expansion	63
2.14	Special Built-In Utilities.....	64

	<i>break</i>	65
	<i>colon</i>	67
	<i>continue</i>	69
	<i>dot</i>	71
	<i>eval</i>	73
	<i>exec</i>	75
	<i>exit</i>	77
	<i>export</i>	79
	<i>readonly</i>	82
	<i>return</i>	84
	<i>set</i>	86
	<i>shift</i>	92
	<i>times</i>	94
	<i>trap</i>	96
	<i>unset</i>	99
Chapter 3	Batch Environment Services	101
3.1	General Concepts	101
3.1.1	Batch Client-Server Interaction	101
3.1.2	Batch Queues	101
3.1.3	Batch Job Creation	102
3.1.4	Batch Job Tracking	102
3.1.5	Batch Job Routing	102
3.1.6	Batch Job Execution	103
3.1.7	Batch Job Exit	103
3.1.8	Batch Job Abort	103
3.1.9	Batch Authorization	103
3.1.10	Batch Administration	104
3.1.11	Batch Notification	104
3.2	Batch Services	104
3.2.1	Batch Job States	105
3.2.2	Deferred Batch Services	106
3.2.2.1	Batch Job Execution	106
3.2.2.2	Batch Job Routing	113
3.2.2.3	Batch Job Exit	113
3.2.2.4	Batch Server Restart	114
3.2.2.5	Batch Job Abort	114
3.2.3	Requested Batch Services	115
3.2.3.1	Delete Batch Job Request	115
3.2.3.2	Hold Batch Job Request	116
3.2.3.3	Batch Job Message Request	116
3.2.3.4	Batch Job Status Request	117
3.2.3.5	Locate Batch Job Request	117
3.2.3.6	Modify Batch Job Request	117
3.2.3.7	Move Batch Job Request	118
3.2.3.8	Queue Batch Job Request	118
3.2.3.9	Batch Queue Status Request	119
3.2.3.10	Release Batch Job Request	119

3.2.3.11	Rerun Batch Job Request	120
3.2.3.12	Select Batch Jobs Request	120
3.2.3.13	Server Shutdown Request.....	120
3.2.3.14	Server Status Request.....	121
3.2.3.15	Signal Batch Job Request	121
3.2.3.16	Track Batch Job Request	121
3.3	Common Behavior for Batch Environment Utilities	122
3.3.1	Batch Job Identifier	122
3.3.2	Destination	123
3.3.3	Multiple Keyword-Value Pairs	123
Chapter 4	Utilities.....	125
	Index.....	1081
List of Figures		
4-1	pax Format Archive Example.....	713
List of Tables		
1-1	Actions when Creating a File that Already Exists	5
1-2	Selected ISO C Standard Operators and Control Flow Keywords.....	8
1-3	Utility Limit Minimum Values.....	17
1-4	Symbolic Utility Limits.....	18
1-5	Regular Built-In Utilities	28
3-1	Batch Utilities.....	101
3-2	Environment Variable Summary	105
3-3	Next State Table	107
3-4	Results / Output Table	108
3-5	Batch Services Summary	115
4-1	Expressions in Decreasing Precedence in <i>awk</i>	156
4-2	Escape Sequences in <i>awk</i>	162
4-3	Operators in <i>bc</i>	198
4-4	Programming Environments: Type Sizes	215
4-5	Programming Environments: <i>c99</i> and <i>cc</i> Arguments.....	216
4-6	ASCII to EBCDIC Conversion.....	305
4-7	ASCII to IBM EBCDIC Conversion.....	306
4-8	File Utility Output Strings	446
4-9	Table Size Declarations in <i>lex</i>	538
4-10	Escape Sequences in <i>lex</i>	540
4-11	ERE Precedence in <i>lex</i>	541
4-12	Named Characters in <i>od</i>	682
4-13	ustar Header Block	718
4-14	ustar <i>mode</i> Field	719
4-15	Octet-Oriented <i>cpio</i> Archive Entry	721
4-16	Values for <i>cpio</i> <i>c_mode</i> Field	722
4-17	Variable Names and Default Headers in <i>ps</i>	755

4-18	Environment Variable Values (Utilities)	804
4-19	Control Character Names in <i>stty</i>	889
4-20	Circumflex Control Characters in <i>stty</i>	889
4-21	uuencode Base64 Values.....	973
4-22	Internal Limits in <i>yacc</i>	1075

Foreword

Structure of the Standard

This standard was originally developed by the Austin Group, a joint working group of members of the IEEE, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1, as one of the four volumes of IEEE Std 1003.1-2001. The standard was approved by ISO and IEC and published in four parts, correlating to the original volumes.

A mapping of the parts to the volumes is shown below:

ISO/IEC 9945 Part	IEEE Std 1003.1 Volume	Description
9945-1	Base Definitions	Includes general terms, concepts, and interfaces common to all parts of ISO/IEC 9945, including utility conventions and C-language header definitions.
9945-2	System Interfaces	Includes definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery.
9945-3	Shell and Utilities	Includes definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs.
9945-4	Rationale	Includes extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of ISO/IEC 9945 and why features were included or discarded by the standard developers.

All four parts comprise the entire standard, and are intended to be used together to accommodate significant internal referencing among them. POSIX-conforming systems are required to support all four parts.

Introduction

Note: This introduction is not part of IEEE Std 1003.1-2001, Standard for Information Technology — Portable Operating System Interface (POSIX).

This standard has been jointly developed by the IEEE and The Open Group. It is simultaneously an IEEE Standard, an ISO/IEC Standard, and an Open Group Technical Standard.

The Austin Group

This standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.³ The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group has been to revise, combine, and update the following standards: ISO/IEC 9945-1, ISO/IEC 9945-2, IEEE Std 1003.1, IEEE Std 1003.2, and the Base Specifications of The Open Group Single UNIX Specification.

After two initial meetings, an agreement was signed in July 1999 between The Open Group and the Institute of Electrical and Electronics Engineers (IEEE), Inc., to formalize the project with the first draft of the revised specifications being made available at the same time. Under this agreement, The Open Group and IEEE agreed to share joint copyright of the resulting work. The Open Group has provided the chair and secretariat for the Austin Group.

The base document for the revision was The Open Group's Base volumes of its Single UNIX Specification, Version 2. These were selected since they were a superset of the existing POSIX.1 and POSIX.2 specifications and had some organizational aspects that would benefit the audience for the new revision.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group's Technical Standard designation, and an ISO/IEC designation. This set of specifications forms the core of the Single UNIX Specification, Version 3.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see <http://www.opengroup.org/austin>.

3. The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

Background

The developers of this standard represent a cross section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, this standard describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application writers to write portable applications—it was developed with that goal in mind—it has been designated POSIX,⁴ an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of standards.

Audience

The intended audience for this standard is all persons concerned with an industry-wide standard operating system based on the UNIX system. This includes at least four groups of people:

1. Persons buying hardware and software systems
2. Persons managing companies that are deciding on future corporate computing directions
3. Persons implementing operating systems, and especially
4. Persons developing applications where portability is an objective

Purpose

Several principles guided the development of this standard:

- Application-Oriented

The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. This standard codifies the common, existing definition of the UNIX system.

- Interface, Not Implementation

This standard defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.

4. The name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

- Source, Not Object, Portability

This standard has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. This standard does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.

- The C Language

The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.

- No Superuser, No System Administration

There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in this standard. This standard is also not concerned with hardware constraints or system maintenance.

- Minimal Interface, Minimally Defined

In keeping with the historical design principles of the UNIX system, the mandatory core facilities of this standard have been kept as minimal as possible. Additional capabilities have been added as optional extensions.

- Broadly Implementable

The developers of this standard endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:

1. All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
2. Compatible systems that are not derived from the original UNIX system code
3. Emulations hosted on entirely different operating systems
4. Networked systems
5. Distributed systems
6. Systems running on a broad range of hardware

No direct references to this goal appear in this standard, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations

When the original version of IEEE Std 1003.1 was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and the subsequent revisions and addenda to all of them have consolidated this consensus, and this revision reflects the significantly increased level of consensus arrived at since the original versions. The earlier standards and their modifications specified a number of areas where consensus had not been reached before, and these are now reflected in this revision. The authors of the original versions tried, as much as possible, to follow the principles below

when creating new specifications:

1. By standardizing an interface like one in an historical implementation; for example, directories
2. By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended *tar* format defined in the *pax* utility
3. By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the *sigaction()* function

This revision tries to minimize the number of changes required to implementations which conform to the earlier versions of the approved standards to bring them into conformance with the current standard. Specifically, the scope of this work excluded doing any “new” work, but rather collecting into a single document what had been spread across a number of documents, and presenting it in what had been proven in practice to be a more effective way. Some changes to prior conforming implementations were unavoidable, primarily as a consequence of resolving conflicts found in prior revisions, or which became apparent when bringing the various pieces together.

However, since it references the 1999 version of the ISO C standard, and no longer supports “Common Usage C”, there are a number of unavoidable changes. Applications portability is similarly affected.

This standard is specifically not a codification of a particular vendor’s product.

It should be noted that implementations will have different kinds of extensions. Some will reflect “historical usage” and will be preserved for execution of pre-existing applications. These functions should be considered “obsolescent” and the standard functions used for new applications. Some extensions will represent functions beyond the scope of this standard. These need to be used with careful management to be able to adapt to future extensions of this standard and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code

A goal of this standard was to minimize additional work for the developers of applications. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

This Standard

This standard defines the Portable Operating System Interface (POSIX) requirements and consists of the following volumes:

- Base Definitions
- Shell and Utilities (this volume)
- System Interfaces
- Rationale (Informative)

This Volume

The Shell and Utilities volume describes the commands and utilities offered to application programs on POSIX-conformant systems. Readers are expected to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards. It also describes the defaults used by the utility descriptions in Chapter 4.
- Chapter 2 describes the command language used in POSIX-conformant systems.
- Chapter 3 describes a set of services and utilities that are implemented on systems supporting the Batch Environment Services and Utilities option.
- Chapter 4 consists of reference pages for all utilities available on POSIX-conformant systems.

Comprehensive references are available in the index.

Typographical Conventions

The following typographical conventions are used throughout this standard. In the text, this standard is referred to as IEEE Std 1003.1-2001, which is technically identical to The Open Group Base Specifications, Issue 6.

The typographical conventions listed here are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this standard.

Reference	Example	Notes
C-Language Data Structure	aiocb	
C-Language Data Structure Member	<i>aio_lio_opcode</i>	
C-Language Data Type	long	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	
C-Language Function Argument	<i>arg1</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	<sys/stat.h>	
C-Language Keyword	return	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	INET_ADDRSTRLEN	
C-Language Preprocessing Directive	#define	
Commands within a Utility	a, c	
Conversion Specification, Specifier / Modifier Character	%A, g, E	1
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	Hello, World	
Filename	/tmp	
Literal Character	'c', '\r', '\'	2
Literal String	"abcde"	2
Optional Items in Utility Syntax	[]	
Parameter	<i><directory pathname></i>	
Special Character	<i><newline></i>	3

Reference	Example	Notes
Symbolic Constant	<code>_POSIX_VDISABLE</code>	4
Symbolic Limit, Configuration Value	<code>{LINE_MAX}</code>	
Syntax	<code>#include <sys/stat.h></code>	5
User Input and Example Code	<code>echo Hello, World</code>	
Utility Name	<code>awk</code>	
Utility Operand	<code>file_name</code>	
Utility Option	<code>-c</code>	
Utility Option with Option-Argument	<code>-w width</code>	

Notes:

1. Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf* and *fscanf* formatting functions.
2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. For literal characters, `'\'` (or any of the other sequences such as `''`) is the same as the C constant `'\\'` (or `'\''`).
3. The style selected for some of the special characters, such as `<newline>`, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters `<tab>` or `<newline>`.
4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
5. Brackets shown in this font, `" [ ] "`, are part of the syntax and do *not* indicate optional items. In syntax the `' | '` symbol is used to separate alternatives, and ellipses (`" . . . "`) are used to show that additional arguments are optional.

Shading is used to identify extensions and options; see Section 1.8.1 (on page 9).

Footnotes and notes within the body of the normative text are for information only (informative).

Informative sections (such as Rationale, Change History, Application Usage, and so on) are denoted by continuous shading bars in the margins.

Ranges of values are indicated with parentheses or brackets as follows:

- (a,b) means the range of all values from a to b , including neither a nor b .
- $[a,b]$ means the range of all values from a to b , including a and b .
- $[a,b)$ means the range of all values from a to b , including a , but not b .
- $(a,b]$ means the range of all values from a to b , including b , but not a .

Participants

IEEE Std 1003.1-2001 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/SC22 WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair

Donald W. Cragun, Organizational Representative, IEEE PASC

Nicholas Stoughton, Organizational Representative, ISO/SC22 WG15

Mark Brown, Organizational Representative, The Open Group

Cathy Hughes, Technical Editor

Austin Group Technical Reviewers

Peter Anvin	Michael Gonzalez	Sandra O'Donnell
Bouazza Bachar	Joseph M. Gwinn	Frank Prindle
Theodore P. Baker	Jon Hitchcock	Curtis Royster Jr.
Walter Briscoe	Yvette Ho Sang	Glen Seeds
Mark Brown	Cathy Hughes	Keld Jorn Simonsen
Dave Butenhof	Lowell G. Johnson	Raja Srinivasan
Geoff Clare	Andrew Josey	Nicholas Stoughton
Donald W. Cragun	Michael Kavanaugh	Donn S. Terry
Lee Damico	David Korn	Fred Tydeman
Ulrich Drepper	Marc Aurele La France	Peter Van Der Veen
Paul Eggert	Jim Meyering	James Youngman
Joanna Farley	Gary Miller	Jim Zepeda
Clive D.W. Feather	Finnbarr P. Murphy	Jason Zions
Andrew Gollan	Joseph S. Myers	

Austin Group Working Group Members

Harold C. Adams	Michael Gonzalez	Sandra O'Donnell
Peter Anvin	Karen D. Gordon	Frank Prindle
Pierre-Jean Arcos	Joseph M. Gwinn	Francois Riche
Jay Ashford	Steven A. Haaser	John D. Riley
Bouazza Bachar	Charles E. Hammons	Andrew K. Roach
Theodore P. Baker	Chris J. Harding	Helmut Roth
Robert Barned	Barry Hedquist	Jaideep Roy
Joel Berman	Vincent E. Henley	Curtis Royster Jr.
David J. Blackwood	Karl Heubaum	Stephen C. Schwarm
Shirley Bockstahler-Brandt	Jon Hitchcock	Glen Seeds
James Bottomley	Yvette Ho Sang	Richard Seibel
Walter Briscoe	Niklas Holsti	David L. Shroads Jr.
Andries Brouwer	Thomas Hosmer	W. Olin Sibert
Mark Brown	Cathy Hughes	Keld Jorn Simonsen
Eric W. Burger	Jim D. Isaak	Curtis Smith
Alan Burns	Lowell G. Johnson	Raja Srinivasan
Andries Brouwer	Michael B. Jones	Nicholas Stoughton
Dave Butenhof	Andrew Josey	Marc J. Teller
Keith Chow	Michael J. Karels	Donn S. Terry
Geoff Clare	Michael Kavanaugh	Fred Tydeman
Donald W. Cragun	David Korn	Mark-Rene Uchida
Lee Damico	Steven Kramer	Scott A. Valcourt
Juan Antonio De La Puente	Thomas M. Kurihara	Peter Van Der Veen
Ming De Zhou	Marc Aurele La France	Michael W. Vannier
Steven J. Dovich	C. Douglass Locke	Eric Vought
Richard P. Draves	Nick Maclaren	Frederick N. Webb
Ulrich Drepper	Roger J. Martin	Paul A.T. Wolfgang
Paul Eggert	Craig H. Meyer	Garrett A. Wollman
Philip H. Enslow	Jim Meyering	James Youngman
Joanna Farley	Gary Miller	Oren Yuen
Clive D.W. Feather	Finnbarr P. Murphy	Janusz Zalewski
Pete Forman	Joseph S. Myers	Jim Zepeda
Mark Funkenhauser	John Napier	Jason Zions
Lois Goldthwaite	Peter E. Obermayer	
Andrew Gollan	James T. Oblinger	

The Open Group

When The Open Group approved the Base Specifications, Issue 6 on 12 September 2001, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair

Finnbarr P. Murphy, Vice-Chair

Mark Brown, Austin Group Liaison

Cathy Hughes, Technical Editor

Base Working Group Members

Bouazza Bachar

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Joanna Farley

Andrew Gollan

Karen D. Gordon

Gary Miller

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair

Joseph M. Gwinn, Vice-Chair

Jay Ashford, Functional Chair

Andrew Josey, Functional Chair

Curtis Royster Jr., Functional Chair

Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001. Balloters may have voted for approval, disapproval, or abstention:

Harold C. Adams

Pierre-Jean Arcos

Jay Ashford

Theodore P. Baker

Robert Barned

David J. Blackwood

Shirley Bockstahler-Brandt

James Bottomley

Mark Brown

Eric W. Burger

Alan Burns

Dave Butenhof

Keith Chow

Donald W. Cragun

Juan Antonio De La Puente

Ming De Zhou

Steven J. Dovich

Richard P. Draves

Philip H. Enslow

Michael Gonzalez

Karen D. Gordon

Joseph M. Gwinn

Steven A. Haaser

Charles E. Hammons

Chris J. Harding

Barry Hedquist

Vincent E. Henley

Karl Heubaum

Niklas Holsti

Thomas Hosmer

Jim D. Isaak

Lowell G. Johnson

Michael B. Jones

Andrew Josey

Michael J. Karels

Steven Kramer

Thomas M. Kurihara

C. Douglass Locke

Roger J. Martin

Craig H. Meyer

Finnbarr P. Murphy

John Napier

Peter E. Obermayer

James T. Oblinger

Frank Prindle

Francois Riche

John D. Riley

Andrew K. Roach

Helmut Roth

Jaideep Roy

Curtis Royster Jr.

Stephen C. Schwarm

Richard Seibel

David L. Shroads Jr.

W. Olin Sibert

Keld Jorn Simonsen

Nicholas Stoughton

Donn S. Terry

Mark-Rene Uchida

Scott A. Valcourt

Michael W. Vannier

Frederick N. Webb

Paul A.T. Wolfgang

Oren Yuen

Janusz Zalewski

The following organizational representative voted on this standard:

Andrew Josey, X/Open Company Ltd.

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, it had the following membership:

Donald N. Heirman, Chair

James T. Carlo, Vice-Chair

Judith Gorman, Secretary

Satish K. Aggarwal

Mark D. Bowman

Gary R. Engmann

Harold E. Epstein

H. Landis Floyd

Jay Forster*

Howard M. Frazier

Ruben D. Garzon

James H. Gurney

Richard J. Holleman

Lowell G. Johnson

Robert J. Kennelly

Joseph L. Koepfinger*

Peter H. Lips

L. Bruce McClung

Daleep C. Mohla

James W. Moore

Robert F. Munzner

Ronald C. Petersen

Gerald H. Peterson

John B. Posey

Gary S. Robinson

Akio Tojo

Donald W. Zipse

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative

Donald R. Volzka, TAB Representative

Yvette Ho Sang, **Don Messina**, **Savoula Amanatidis**, IEEE Project Editors

* Member Emeritus

IEEE Std 1003.1-2001/Cor 1-2002 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/IEC JTC 1/SC22/WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair

Donald W. Cragun, Organizational Representative, IEEE PASC

Nicholas Stoughton, Organizational Representative, ISO/IEC JTC 1/SC22/WG15

Mark Brown, Organizational Representative, The Open Group

Cathy Fox, Technical Editor

Austin Group Technical Reviewers

Theodore P. Baker

Julian Blake

Andries Brouwer

Mark Brown

Dave Butenhof

Geoff Clare

Donald W. Cragun

Ken Dawson

Ulrich Drepper

Larry Dwyer

Paul Eggert

Joanna Farley

Clive D.W. Feather

Cathy Fox

Mark Funkenhauser

Lois Goldthwaite

Andrew Gollan

Michael Gonzalez

Bruno Haible

Ben Harris

Jon Hitchcock

Andreas Jaeger

Andrew Josey

Jonathan Lennox

Nick Maclaren

Jack McCann

Wilhelm Mueller

Joseph S. Myers

Frank Prindle

Kenneth Raeburn

Tim Robbins

Glen Seeds

Matthew Seitz

Keld Jorn Simonsen

Nicholas Stoughton

Alexander Terekhov

Donn S. Terry

Mike Wilson

Garrett A. Wollman

Mark Ziegast

Austin Group Working Group Members

Harold C. Adams
Alejandro Alonso
Jay Ashford
Theodore P. Baker
David J. Blackwood
Julian Blake
Mitchell Bonnett
Andries Brouwer
Mark Brown
Eric W. Burger
Alan Burns
Dave Butenhof
Keith Chow
Geoff Clare
Luis Cordova
Donald W. Cragun
Dragan Cvetkovic
Lee Damico
Ken Dawson
Jeroen Dekkers
Juan Antonio De La Puente
Steven J. Dovich
Ulrich Drepper
Dr. Sourav Dutta
Larry Dwyer
Paul Eggert
Joanna Farley

Clive D.W. Feather
Yaacov Fenster
Cathy Fox
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan
Michael Gonzalez
Karen D. Gordon
Scott Gudgel
Joseph M. Gwinn
Steven A. Haaser
Bruno Haible
Charles E. Hammons
Bryan Harold
Ben Harris
Barry Hedquist
Karl Heubaum
Jon Hitchcock
Andreas Jaeger
Andrew Josey
Kenneth Lang
Pi-Cheng Law
Jonathan Lennox
Nick Maclaren
Roger J. Martin
Jack McCann
George Miao

Wilhelm Mueller
Finnbarr P. Murphy
Joseph S. Myers
Alexey Neyman
Charles Ngethe
Peter Petrov
Frank Prindle
Vikram Punj
Kenneth Raeburn
Francois Riche
Tim Robbins
Curtis Royster Jr.
Diane Schleicher
Gil Shultz
Stephen C. Schwarm
Glen Seeds
Matthew Seitz
Keld Jorn Simonsen
Doug Stevenson
Nicholas Stoughton
Alexander Terekhov
Donn S. Terry
Mike Wilson
Garrett A. Wollman
Oren Yuen
Mark Ziegast

The Open Group

When The Open Group approved the Base Specifications, Issue 6, Technical Corrigendum 1 on 7 February 2003, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair

Finnbarr P. Murphy, Vice-Chair

Mark Brown, Austin Group Liaison

Cathy Fox, Technical Editor

Base Working Group Members

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Ulrich Drepper

Joanna Farley

Andrew Gollan

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001/Cor 1-2002 on 11 December 2002, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair

Joseph M. Gwinn, Vice-Chair

Jay Ashford, Functional Chair

Andrew Josey, Functional Chair

Curtis Royster Jr., Functional Chair

Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001/Cor 1-2002. Balloters may have voted for approval, disapproval, or abstention:

Alejandro Alonso

Jay Ashford

David J. Blackwood

Julian Blake

Mitchell Bonnett

Mark Brown

Dave Butenhof

Keith Chow

Luis Cordova

Donald W. Cragun

Steven J. Dovich

Dr. Sourav Dutta

Yaacov Fenster

Michael Gonzalez

Scott Gudgel

Charles E. Hammons

Bryan Harold

Barry Hedquist

Karl Heubaum

Lowell G. Johnson

Andrew Josey

Kenneth Lang

Pi-Cheng Law

George Miao

Roger J. Martin

Finnbarr P. Murphy

Charles Ngethe

Peter Petrov

Frank Prindle

Vikram Punj

Francois Riche

Curtis Royster Jr.

Diane Schleicher

Stephen C. Schwarm

Gil Shultz

Nicholas Stoughton

Donn S. Terry

Oren Yuen

Juan A. de la Puente

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 / Cor 1-2002 on 11 December 2002, the membership was as follows:

James T. Carlo, Chair

James H. Gurney, Vice-Chair

Judith Gorman, Secretary

Sid Bennett

H. Stephen Berger

Clyde R. Camp

Richard DeBlasio

Harold E. Epstein

Julian Forster*

Howard M. Frazier

Toshio Fukuda

Arnold M. Greenspan

Raymond Hapeman

Donald M. Heirman

Richard H. Hulett

Lowell G. Johnson

Joseph L. Koepfinger*

Peter H. Lips

Nader Mehravari

Daleep C. Mohla

William J. Moylan

Malcolm V. Thaden

Geoffrey O. Thompson

Howard L. Wolfman

Don Wright

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative

Satish K. Aggarwal, NRC Representative

Savoula Amanatidis, IEEE Standards Managing Editor

* Member Emeritus

Trademarks

The following information is given for the convenience of users of this standard and does not constitute endorsement of these products by The Open Group or the IEEE. There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

1003.1[™] is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

AIX[®] is a registered trademark of IBM Corporation.

AT&T[®] is a registered trademark of AT&T in the U.S.A. and other countries.

BSD[™] is a trademark of the University of California, Berkeley, U.S.A.

Hewlett-Packard[®], HP[®], and HP-UX[®] are registered trademarks of Hewlett-Packard Company.

IBM[®] is a registered trademark of International Business Machines Corporation.

The Open Group and Boundaryless Information Flow are trademarks and UNIX is a registered trademark of The Open Group in the United States and other countries. All other trademarks are the property of their respective owners.

POSIX[®] is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Sun[®] and Sun Microsystems[®] are registered trademarks of Sun Microsystems, Inc.

/usr/group[®] is a registered trademark of UniForum, the International Network of UNIX System Users.

Acknowledgements

The contributions of the following organizations to the development of IEEE Std 1003.1-2001 are gratefully acknowledged:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.
- The SC22 WG14 Committees.

This standard was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO SC22 WG15.

Referenced Documents

Normative References

Normative references for this standard are defined in the Base Definitions volume.

Informative References

The following documents are referenced in this standard:

1984 /usr/group Standard

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

Almasi and Gottlieb

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

ANSI C

American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.226-1994

American National Standard for Information Systems: Standard X3.226-1994, Programming Language Common LISP.

Brawer

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989, ISBN: 0-12-128470-0.

DeRemer and Pennello Article

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets*, SigPlan Notices, Volume 15, No. 8, August 1979.

Draft ANSI X3J11.1

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-1

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

FIPS 151-2

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

HP-UX Manual

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

IEC 60559: 1989

IEC 60559: 1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously designated IEC 559: 1989).

IEEE Std 754-1985

IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic.

IEEE Std 854-1987

IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.

- IEEE Std 1003.9-1992
IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.
- IETF RFC 791
Internet Protocol, Version 4 (IPv4), September 1981.
- IETF RFC 819
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982.
- IETF RFC 822
Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982.
- IETF RFC 919
Broadcasting Internet Datagrams, J. Mogul, October 1984.
- IETF RFC 920
Domain Requirements, J. Postel, J. Reynolds, October 1984.
- IETF RFC 921
Domain Name System Implementation Schedule, J. Postel, October 1984.
- IETF RFC 922
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984.
- IETF RFC 1034
Domain Names — Concepts and Facilities, P. Mockapetris, November 1987.
- IETF RFC 1035
Domain Names — Implementation and Specification, P. Mockapetris, November 1987.
- IETF RFC 1123
Requirements for Internet Hosts — Application and Support, R. Braden, October 1989.
- IETF RFC 1886
DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson, December 1995.
- IETF RFC 2045
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996.
- IETF RFC 2181
Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997.
- IETF RFC 2373
Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998.
- IETF RFC 2460
Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998.
- Internationalisation Guide
Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.
- ISO C (1990)
ISO/IEC 9899:1990, Programming Languages — C, including Amendment 1:1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).

ISO 2375:1985

ISO 2375:1985, Data Processing — Procedure for Registration of Escape Sequences.

ISO 8652:1987

ISO 8652:1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).

ISO/IEC 1539:1990

ISO/IEC 1539:1990, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

ISO/IEC 4873:1991

ISO/IEC 4873:1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.

ISO/IEC 6429:1992

ISO/IEC 6429:1992, Information Technology — Control Functions for Coded Character Sets.

ISO/IEC 6937:1994

ISO/IEC 6937:1994, Information Technology — Coded Character Set for Text Communication — Latin Alphabet.

ISO/IEC 8802-3:1996

ISO/IEC 8802-3:1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

ISO/IEC 8859

ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

Part 1: Latin Alphabet No. 1

Part 2: Latin Alphabet No. 2

Part 3: Latin Alphabet No. 3

Part 4: Latin Alphabet No. 4

Part 5: Latin/Cyrillic Alphabet

Part 6: Latin/Arabic Alphabet

Part 7: Latin/Greek Alphabet

Part 8: Latin/Hebrew Alphabet

Part 9: Latin Alphabet No. 5

Part 10: Latin Alphabet No. 6

Part 13: Latin Alphabet No. 7

Part 14: Latin Alphabet No. 8

Part 15: Latin Alphabet No. 9

ISO POSIX-1:1996

ISO/IEC 9945-1:1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Std 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.

ISO POSIX-2:1993

ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2-1992, as amended by ANSI/IEEE Std 1003.2a-1992).

Issue 1

X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

Issue 2

X/Open Portability Guide, January 1987:

- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
- Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)

Issue 3

X/Open Specification, 1988, 1989, February 1992:

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

Issue 4

CAE Specification, July 1992, published by The Open Group:

- System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
- Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
- System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

Issue 4, Version 2

CAE Specification, August 1994, published by The Open Group:

- System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

Issue 5

Technical Standard, February 1997, published by The Open Group:

- System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

Knuth Article

Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control, Volume 8, No. 6, October 1965.

KornShell

Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming Language*, March 1995, Prentice Hall.

MSE Working Draft

Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

POSIX.0: 1995

IEEE Std 1003.0-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical to ISO/IEC TR 14252).

POSIX.1: 1988

IEEE Std 1003.1-1988, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1: 1990

IEEE Std 1003.1-1990, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1a

P1003.1a, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — (C Language) Amendment.

POSIX.1d: 1999

IEEE Std 1003.1d-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 4: Additional Realtime Extensions [C Language].

POSIX.1g: 2000

IEEE Std 1003.1g-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 6: Protocol-Independent Interfaces (PII).

POSIX.1j: 2000

IEEE Std 1003.1j-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 5: Advanced Realtime Extensions [C Language].

POSIX.1q: 2000

IEEE Std 1003.1q-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 7: Tracing [C Language].

POSIX.2b

P1003.2b, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment.

POSIX.2d:-1994

IEEE Std 1003.2d-1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.

POSIX.13:-1998

IEEE Std 1003.13:1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.

Sarwate Article

Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications of the ACM, Volume 30, No. 8, August 1988.

Sprunt, Sha, and Lehoczky

Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*, The Journal of Real-Time Systems, Volume 1, 1989, Pages 27-60.

SVID, Issue 1

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.

SVID, Issue 2

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.

SVID, Issue 3

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.

The AWK Programming Language

Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.

UNIX Programmer's Manual

American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.

XNS, Issue 4

CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.

XNS, Issue 5

CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XNS, Issue 5.2

Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.

X/Open Curses, Issue 4, Version 2

CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.

Yacc

Yacc: Yet Another Compiler Compiler, Stephen C. Johnson, 1978.

Source Documents

Parts of the following documents were used to create the base documents for this standard:

AIX 3.2 Manual

AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).

OSF/1

OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).

OSF AES

Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

System V Release 2.0

- UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).
- UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).

System V Release 4.2

Operating System API Reference, UNIX SVR4.2 (1992) (ISBN: 0-13-017658-3).

Introduction

1.1 Scope

The scope of IEEE Std 1003.1-2001 is described in the Base Definitions volume of IEEE Std 1003.1-2001.

1.2 Conformance

Conformance requirements for IEEE Std 1003.1-2001 are defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance.

1.3 Normative References

Normative references for IEEE Std 1003.1-2001 are defined in the Base Definitions volume of IEEE Std 1003.1-2001.

1.4 Change History

Change history is described in the Rationale (Informative) volume of IEEE Std 1003.1-2001, and in the CHANGE HISTORY section of reference pages.

1.5 Terminology

This section appears in the Base Definitions volume of IEEE Std 1003.1-2001, but is repeated here for convenience:

For the purposes of IEEE Std 1003.1-2001, the following terminology definitions apply:

can

Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to IEEE Std 1003.1-2001. An application can rely on the existence of the feature or behavior.

implementation-defined

Describes a value or behavior that is not defined by IEEE Std 1003.1-2001 but is selected by an implementor. The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior so that it can be used correctly by an application.

legacy

Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable

applications. New applications should use alternative means of obtaining equivalent functionality.

may

Describes a feature or behavior that is optional for an implementation that conforms to IEEE Std 1003.1-2001. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

shall

For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or behavior that is mandatory. An application can rely on the existence of the feature or behavior.

For an application or user, describes a behavior that is mandatory.

should

For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or behavior that is recommended but not mandatory. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

For an application, describes a feature or behavior that is recommended programming practice for optimum portability.

undefined

Describes the nature of a value or behavior not defined by IEEE Std 1003.1-2001 which results from use of an invalid program construct or invalid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

unspecified

Describes the nature of a value or behavior not specified by IEEE Std 1003.1-2001 which results from use of a valid program construct or valid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

1.6 Definitions

Concepts and definitions are defined in the Base Definitions volume of IEEE Std 1003.1-2001.

1.7 Relationship to Other Documents

1.7.1 System Interfaces

This subsection describes some of the features provided by the System Interfaces volume of IEEE Std 1003.1-2001 that are assumed to be globally available on all systems conforming to this volume of IEEE Std 1003.1-2001. This subsection does not attempt to detail all of the features defined in the System Interfaces volume of IEEE Std 1003.1-2001 that are required by all of the utilities defined in this volume of IEEE Std 1003.1-2001; the utility and function descriptions point out additional functionality required to provide the corresponding specific features needed by each.

The following subsections describe frequently used concepts. Many of these concepts are described in the Base Definitions volume of IEEE Std 1003.1-2001. Utility and function description statements override these defaults when appropriate.

1.7.1.1 Process Attributes

The following process attributes, as described in the System Interfaces volume of IEEE Std 1003.1-2001, are assumed to be supported for all processes in this volume of IEEE Std 1003.1-2001:

Controlling Terminal	Real Group ID
Current Working Directory	Real User ID
Effective Group ID	Root Directory
Effective User ID	Saved Set-Group-ID
File Descriptors	Saved Set-User-ID
File Mode Creation Mask	Session Membership
Process Group ID	Supplementary Group IDs
Process ID	

A conforming implementation may include additional process attributes.

1.7.1.2 Concurrent Execution of Processes

The following functionality of the *fork()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 shall be available on all systems conforming to this volume of IEEE Std 1003.1-2001:

1. Independent processes shall be capable of executing independently without either process terminating.
2. A process shall be able to create a new process with all of the attributes referenced in Section 1.7.1.1, determined according to the semantics of a call to the *fork()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 followed by a call in the child process to one of the *exec* functions defined in the System Interfaces volume of IEEE Std 1003.1-2001.

105 1.7.1.3 *File Access Permissions*

106 The file access control mechanism described by the Base Definitions volume of
 107 IEEE Std 1003.1-2001, Section 4.4, File Access Permissions shall apply to all files on an
 108 implementation conforming to this volume of IEEE Std 1003.1-2001.

109 1.7.1.4 *File Read, Write, and Creation*

110 If a file that does not exist is to be written, it shall be created as described below, unless the
 111 utility description states otherwise.

112 When a file that does not exist is created, the following features defined in the System Interfaces
 113 volume of IEEE Std 1003.1-2001 shall apply unless the utility or function description states
 114 otherwise:

- 115 1. The user ID of the file shall be set to the effective user ID of the calling process.
- 116 2. The group ID of the file shall be set to the effective group ID of the calling process or the
 117 group ID of the directory in which the file is being created.
- 118 3. If the file is a regular file, the permission bits of the file shall be set to:

119 S_IROTH | S_IWOTH | S_IRGRP | S_IWGRP | S_IRUSR | S_IWUSR

120 (see the description of *File Modes* in the Base Definitions volume of IEEE Std 1003.1-2001,
 121 Chapter 13, Headers, <**sys/stat.h**>) except that the bits specified by the file mode creation
 122 mask of the process shall be cleared. If the file is a directory, the permission bits shall be set
 123 to:

124 S_IRWXU | S_IRWXG | S_IRWXO

125 except that the bits specified by the file mode creation mask of the process shall be cleared.

- 126 4. The *st_atime*, *st_ctime*, and *st_mtime* fields of the file shall be updated as specified in the
 127 System Interfaces volume of IEEE Std 1003.1-2001, Section 2.5, Standard I/O Streams.
- 128 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length
 129 zero.
- 130 6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2_SYMLINKS}
 131 variable is in effect for the directory in which the symbolic link would be created.
- 132 7. Unless otherwise specified, the file created shall be a regular file.

133 When an attempt is made to create a file that already exists, the utility shall take the action
 134 indicated in Table 1-1 (on page 5) corresponding to the type of the file the utility is trying to
 135 create and the type of the existing file, unless the utility description states otherwise.

Table 1-1 Actions when Creating a File that Already Exists

Existing Type	New Type											Function Creating New
	B	C	D	F	L	M	P	Q	R	S	T	
A <i>fattach()</i> -ed STREAM	F	F	F	F	F	—	—	—	OF	—	—	N / A
B Block Special	F	F	F	F	F	—	—	—	OF	—	—	<i>mknod()</i> **
C Character Special	F	F	F	F	F	—	—	—	OF	—	—	<i>mknod()</i> **
D Directory	F	F	F	F	F	—	—	—	F	—	—	<i>mkdir()</i>
F FIFO Special File	F	F	F	F	F	—	—	—	O	—	—	<i>mkfifo()</i>
L Symbolic Link	F	F	F	F	F	—	—	—	FL	—	—	<i>symlink()</i>
M Shared Memory	F	F	F	F	F	—	—	—	—	—	—	<i>shm_open()</i>
P Semaphore	F	F	F	F	F	—	—	—	—	—	—	<i>sem_open()</i>
Q Message Queue	F	F	F	F	F	—	—	—	—	—	—	<i>mq_open()</i>
R Regular File	F	F	F	F	F	—	—	—	RF	—	—	<i>open()</i>
S Socket	F	F	F	F	F	—	—	—	—	—	—	<i>bind()</i>
T Typed Memory	F	F	F	F	F	—	—	—	—	—	—	*

The following codes are used in Table 1-1:

F Fail. The attempt to create the new file shall fail and the utility shall either continue with its operation or exit immediately with a non-zero exit status, depending on the description of the utility.

FL Follow link. Unless otherwise specified, the symbolic link shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.

O Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:

1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.
2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.

OF The named file shall be opened with the consequences defined for that file type.

RF Regular file. When attempting to create a regular file, and the existing file is a regular file:

1. The user ID, group ID, and permission bits of the file shall not be changed.
2. The file shall be truncated to zero length.
3. The *st_ctime* and *st_mtime* fields shall be marked for update.

— The effect is implementation-defined unless specified by the utility description.

***** There is no portable way to create a file of this type.

****** Not portable.

When a file is to be appended, the file shall be opened in a manner equivalent to using the *O_APPEND* flag, without the *O_TRUNC* flag, in the *open()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001.

When a file is to be read or written, the file shall be opened with an access mode corresponding to the operation to be performed. If file access permissions deny access, the requested operation shall fail.

179 1.7.1.5 File Removal

180 When a directory that is the root directory or current working directory of any process is
 181 removed, the effect is implementation-defined. If file access permissions deny access, the
 182 requested operation shall fail. Otherwise, when a file is removed:

- 183 1. Its directory entry shall be removed from the file system.
- 184 2. The link count of the file shall be decremented.
- 185 3. If the file is an empty directory (see the Base Definitions volume of IEEE Std 1003.1-2001,
 186 Section 3.143, Empty Directory):
 - 187 a. If no process has the directory open, the space occupied by the directory shall be
 188 freed and the directory shall no longer be accessible.
 - 189 b. If one or more processes have the directory open, the directory contents shall be
 190 preserved until all references to the file have been closed.
- 191 4. If the file is a directory that is not empty, the *st_ctime* field shall be marked for update.
- 192 5. If the file is not a directory:
 - 193 a. If the link count becomes zero:
 - 194 i. If no process has the file open, the space occupied by the file shall be freed and
 195 the file shall no longer be accessible.
 - 196 ii. If one or more processes have the file open, the file contents shall be preserved
 197 until all references to the file have been closed.
 - 198 b. If the link count is not reduced to zero, the *st_ctime* field shall be marked for update.
- 199 6. The *st_ctime* and *st_mtime* fields of the containing directory shall be marked for update.

200 1.7.1.6 File Time Values

201 All files shall have the three time values described by the Base Definitions volume of
 202 IEEE Std 1003.1-2001, Section 4.7, File Times Update.

203 1.7.1.7 File Contents

204 When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of
 205 the data placed in the space pointed to by *buf* when performing the *read()* function calls in the
 206 following operations defined in the System Interfaces volume of IEEE Std 1003.1-2001:

```
207 while (read (fildes, buf, nbytes) > 0)
208     ;
```

209 If the file is indicated by a pathname *pathname*, the file descriptor shall be determined by the
 210 equivalent of the following operation defined in the System Interfaces volume of
 211 IEEE Std 1003.1-2001:

```
212 fildes = open (pathname, O_RDONLY);
```

213 The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data
 214 returned by *read()* would vary with different values, the value shall be one that results in the
 215 most data being returned.

216 If the *read()* function calls would return an error, it is unspecified whether the contents of the file
 217 are considered to include any data from offsets in the file beyond where the error would be
 218 returned.

219 1.7.1.8 *Pathname Resolution*

220 The pathname resolution algorithm, described by the Base Definitions volume of
221 IEEE Std 1003.1-2001, Section 4.11, Pathname Resolution, shall be used by implementations
222 conforming to this volume of IEEE Std 1003.1-2001; see also the Base Definitions volume of
223 IEEE Std 1003.1-2001, Section 4.5, File Hierarchy.

224 1.7.1.9 *Changing the Current Working Directory*

225 When the current working directory (see the Base Definitions volume of IEEE Std 1003.1-2001,
226 Section 3.436, Working Directory) is to be changed, unless the utility or function description
227 states otherwise, the operation shall succeed unless a call to the *chdir()* function defined in the
228 System Interfaces volume of IEEE Std 1003.1-2001 would fail when invoked with the new
229 working directory pathname as its argument.

230 1.7.1.10 *Establish the Locale*

231 The functionality of the *setlocale()* function defined in the System Interfaces volume of
232 IEEE Std 1003.1-2001 shall be available on all systems conforming to this volume of
233 IEEE Std 1003.1-2001; that is, utilities that require the capability of establishing an international
234 operating environment shall be permitted to set the specified category of the international
235 environment.

236 1.7.1.11 *Actions Equivalent to Functions*

237 Some utility descriptions specify that a utility performs actions equivalent to a function defined
238 in the System Interfaces volume of IEEE Std 1003.1-2001. Such specifications require only that
239 the external effects be equivalent, not that any effect within the utility and visible only to the
240 utility be equivalent.

241 1.7.2 **Concepts Derived from the ISO C Standard**

242 Some of the standard utilities perform complex data manipulation using their own procedure
243 and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS
244 sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type
245 conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard,
246 as described in the following sections. Note that there is no requirement that the standard
247 utilities be implemented in any particular programming language.

248 1.7.2.1 *Arithmetic Precision and Operations*

249 Integer variables and constants, including the values of operands and option-arguments, used
250 by the standard utilities listed in this volume of IEEE Std 1003.1-2001 shall be implemented as
251 equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as
252 equivalent to the ISO C standard **double** type. Conversions between types shall be as described
253 in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned
254 by the input to the application.

255 Arithmetic operators and control flow keywords shall be implemented as equivalent to those in
256 the cited ISO C standard section, as listed in Table 1-2 (on page 8).

Table 1-2 Selected ISO C Standard Operators and Control Flow Keywords

Operation	ISO C Standard Equivalent Reference
()	Section 6.5.1, Primary Expressions
postfix ++ postfix --	Section 6.5.2, Postfix Operators
unary + unary - prefix ++ prefix -- ~ ! sizeof()	Section 6.5.3, Unary Operators
* / %	Section 6.5.5, Multiplicative Operators
+ -	Section 6.5.6, Additive Operators
<< >>	Section 6.5.7, Bitwise Shift Operators
<, <= >, >=	Section 6.5.8, Relational Operators
= = !=	Section 6.5.9, Equality Operators
&	Section 6.5.10, Bitwise AND Operator
^	Section 6.5.11, Bitwise Exclusive OR Operator
	Section 6.5.12, Bitwise Inclusive OR Operator
&&	Section 6.5.13, Logical AND Operator
	Section 6.5.14, Logical OR Operator
expr?expr:expr	Section 6.5.15, Conditional Operator
=, *=, /=, %=, +=, -= <<=, >>=, &=, ^=, =	Section 6.5.16, Assignment Operators
if () if () ... else switch ()	Section 6.8.4, Selection Statements
while () do ... while () for ()	Section 6.8.5, Iteration Statements
goto continue break return	Section 6.8.6, Jump Statements

The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5, Expressions, of the ISO C standard.

301 1.7.2.2 Mathematical Functions

302 Any mathematical functions with the same names as those in the following sections of the ISO C
303 standard:

- 304 • Section 7.12, Mathematics, `<math.h>`
- 305 • Section 7.20.2, Pseudo-Random Sequence Generation Functions

306 shall be implemented to return the results equivalent to those returned from a call to the
307 corresponding function described in the ISO C standard.

308 1.8 Portability

309 Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 and functions in
310 the System Interfaces volume of IEEE Std 1003.1-2001 describe functionality that might not be
311 fully portable to systems meeting the requirements for POSIX conformance (see the Base
312 Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance).

313 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in
314 the margin identifies the nature of the option, extension, or warning (see Section 1.8.1). For
315 maximum portability, an application should avoid such functionality.

316 Unless the primary task of a utility is to produce textual material on its standard output,
317 application developers should not rely on the format or content of any such material that may be
318 produced. Where the primary task is to provide such material, but the output format is
319 incompletely specified, the description is marked with the OF margin code and shading.
320 Application developers are warned not to expect that the output of such an interface on one
321 system is any guide to its behavior on another system.

322 1.8.1 Codes

323 Codes and their meanings are listed in the Base Definitions volume of IEEE Std 1003.1-2001, but
324 are repeated here for convenience:

325 ADV **Advisory Information**

326 The functionality described is optional. The functionality described is also an extension to the
327 ISO C standard.

328 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.
329 Where additional semantics apply to a function, the material is identified by use of the ADV
330 margin legend.

331 AIO **Asynchronous Input and Output**

332 The functionality described is optional. The functionality described is also an extension to the
333 ISO C standard.

334 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.
335 Where additional semantics apply to a function, the material is identified by use of the AIO
336 margin legend.

337 BAR **Barriers**

338 The functionality described is optional. The functionality described is also an extension to the
339 ISO C standard.

340 Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
341 Where additional semantics apply to a function, the material is identified by use of the BAR
342 margin legend.

343	BE	Batch Environment Services and Utilities
344		The functionality described is optional.
345		Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
346		Where additional semantics apply to a utility, the material is identified by use of the BE margin
347		legend.
348	CD	C-Language Development Utilities
349		The functionality described is optional.
350		Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
351		Where additional semantics apply to a utility, the material is identified by use of the CD margin
352		legend.
353	CPT	Process CPU-Time Clocks
354		The functionality described is optional. The functionality described is also an extension to the
355		ISO C standard.
356		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
357		Where additional semantics apply to a function, the material is identified by use of the CPT
358		margin legend.
359	CS	Clock Selection
360		The functionality described is optional. The functionality described is also an extension to the
361		ISO C standard.
362		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
363		Where additional semantics apply to a function, the material is identified by use of the CS
364		margin legend.
365	CX	Extension to the ISO C standard
366		The functionality described is an extension to the ISO C standard. Application writers may make
367		use of an extension as it is supported on all IEEE Std 1003.1-2001-conforming systems.
368		With each function or header from the ISO C standard, a statement to the effect that “any
369		conflict is unintentional” is included. That is intended to refer to a direct conflict.
370		IEEE Std 1003.1-2001 acts in part as a profile of the ISO C standard, and it may choose to further
371		constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered
372		conflicts.
373		Where additional semantics apply to a function or header, the material is identified by use of the
374		CX margin legend.
375	FD	FORTTRAN Development Utilities
376		The functionality described is optional.
377		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
378		Where additional semantics apply to a utility, the material is identified by use of the FD margin
379		legend.
380	FR	FORTTRAN Runtime Utilities
381		The functionality described is optional.
382		Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
383		Where additional semantics apply to a utility, the material is identified by use of the FR margin
384		legend.
385	FSC	File Synchronization
386		The functionality described is optional. The functionality described is also an extension to the
387		ISO C standard.

388		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
389		Where additional semantics apply to a function, the material is identified by use of the FSC
390		margin legend.
391	IP6	IPV6
392		The functionality described is optional. The functionality described is also an extension to the
393		ISO C standard.
394		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
395		Where additional semantics apply to a function, the material is identified by use of the IP6
396		margin legend.
397	MC1	Advisory Information and either Memory Mapped Files or Shared Memory Objects
398		The functionality described is optional. The functionality described is also an extension to the
399		ISO C standard.
400		This is a shorthand notation for combinations of multiple option codes.
401		Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
402		Where additional semantics apply to a function, the material is identified by use of the MC1
403		margin legend.
404		Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
405		Notation.
406	MC2	Memory Mapped Files, Shared Memory Objects, or Memory Protection
407		The functionality described is optional. The functionality described is also an extension to the
408		ISO C standard.
409		This is a shorthand notation for combinations of multiple option codes.
410		Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.
411		Where additional semantics apply to a function, the material is identified by use of the MC2
412		margin legend.
413		Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
414		Notation.
415	MC3	Memory Mapped Files, Shared Memory Objects, or Typed Memory Objects
416		The functionality described is optional. The functionality described is also an extension to the
417		ISO C standard.
418		This is a shorthand notation for combinations of multiple option codes.
419		Where applicable, functions are marked with the MC3 margin legend in the SYNOPSIS section.
420		Where additional semantics apply to a function, the material is identified by use of the MC3
421		margin legend.
422		Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
423		Notation.
424	MF	Memory Mapped Files
425		The functionality described is optional. The functionality described is also an extension to the
426		ISO C standard.
427		Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.
428		Where additional semantics apply to a function, the material is identified by use of the MF
429		margin legend.
430	ML	Process Memory Locking
431		The functionality described is optional. The functionality described is also an extension to the

432		ISO C standard.
433		Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
434		Where additional semantics apply to a function, the material is identified by use of the ML
435		margin legend.
436	MLR	Range Memory Locking
437		The functionality described is optional. The functionality described is also an extension to the
438		ISO C standard.
439		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
440		Where additional semantics apply to a function, the material is identified by use of the MLR
441		margin legend.
442	MON	Monotonic Clock
443		The functionality described is optional. The functionality described is also an extension to the
444		ISO C standard.
445		Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
446		Where additional semantics apply to a function, the material is identified by use of the MON
447		margin legend.
448	MPR	Memory Protection
449		The functionality described is optional. The functionality described is also an extension to the
450		ISO C standard.
451		Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
452		Where additional semantics apply to a function, the material is identified by use of the MPR
453		margin legend.
454	MSG	Message Passing
455		The functionality described is optional. The functionality described is also an extension to the
456		ISO C standard.
457		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
458		Where additional semantics apply to a function, the material is identified by use of the MSG
459		margin legend.
460	MX	IEC 60559 Floating-Point Option
461		The functionality described is optional. The functionality described is also an extension to the
462		ISO C standard.
463		Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.
464		Where additional semantics apply to a function, the material is identified by use of the MX
465		margin legend.
466	OB	Obsolescent
467		The functionality described may be withdrawn in a future version of this volume of
468		IEEE Std 1003.1-2001. Strictly Conforming POSIX Applications and Strictly Conforming XSI
469		Applications shall not use obsolescent features.
470		Where applicable, the material is identified by use of the OB margin legend.
471	OF	Output Format Incompletely Specified
472		The functionality described is an XSI extension. The format of the output produced by the utility
473		is not fully specified. It is therefore not possible to post-process this output in a consistent
474		fashion. Typical problems include unknown length of strings and unspecified field delimiters.
475		Where applicable, the material is identified by use of the OF margin legend.

476	OH	Optional Header
477		In the SYNOPSIS section of some interfaces in the System Interfaces volume of
478		IEEE Std 1003.1-2001 an included header is marked as in the following example:
479	OH	<code>#include <sys/types.h></code>
480		<code>#include <grp.h></code>
481		<code>struct group *getgrnam(const char *name);</code>
482		The OH margin legend indicates that the marked header is not required on XSI-conformant
483		systems.
484	PIO	Prioritized Input and Output
485		The functionality described is optional. The functionality described is also an extension to the
486		ISO C standard.
487		Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
488		Where additional semantics apply to a function, the material is identified by use of the PIO
489		margin legend.
490	PS	Process Scheduling
491		The functionality described is optional. The functionality described is also an extension to the
492		ISO C standard.
493		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
494		Where additional semantics apply to a function, the material is identified by use of the PS
495		margin legend.
496	RS	Raw Sockets
497		The functionality described is optional. The functionality described is also an extension to the
498		ISO C standard.
499		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
500		Where additional semantics apply to a function, the material is identified by use of the RS
501		margin legend.
502	RTS	Realtime Signals Extension
503		The functionality described is optional. The functionality described is also an extension to the
504		ISO C standard.
505		Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
506		Where additional semantics apply to a function, the material is identified by use of the RTS
507		margin legend.
508	SD	Software Development Utilities
509		The functionality described is optional.
510		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
511		Where additional semantics apply to a utility, the material is identified by use of the SD margin
512		legend.
513	SEM	Semaphores
514		The functionality described is optional. The functionality described is also an extension to the
515		ISO C standard.
516		Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
517		Where additional semantics apply to a function, the material is identified by use of the SEM
518		margin legend.
519	SHM	Shared Memory Objects
520		The functionality described is optional. The functionality described is also an extension to the

521 ISO C standard.

522 Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.

523 Where additional semantics apply to a function, the material is identified by use of the SHM

524 margin legend.

525 SIO **Synchronized Input and Output**

526 The functionality described is optional. The functionality described is also an extension to the

527 ISO C standard.

528 Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.

529 Where additional semantics apply to a function, the material is identified by use of the SIO

530 margin legend.

531 SPI **Spin Locks**

532 The functionality described is optional. The functionality described is also an extension to the

533 ISO C standard.

534 Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.

535 Where additional semantics apply to a function, the material is identified by use of the SPI

536 margin legend.

537 SPN **Spawn**

538 The functionality described is optional. The functionality described is also an extension to the

539 ISO C standard.

540 Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.

541 Where additional semantics apply to a function, the material is identified by use of the SPN

542 margin legend.

543 SS **Process Sporadic Server**

544 The functionality described is optional. The functionality described is also an extension to the

545 ISO C standard.

546 Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.

547 Where additional semantics apply to a function, the material is identified by use of the SS

548 margin legend.

549 TCT **Thread CPU-Time Clocks**

550 The functionality described is optional. The functionality described is also an extension to the

551 ISO C standard.

552 Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.

553 Where additional semantics apply to a function, the material is identified by use of the TCT

554 margin legend.

555 TEF **Trace Event Filter**

556 The functionality described is optional. The functionality described is also an extension to the

557 ISO C standard.

558 Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.

559 Where additional semantics apply to a function, the material is identified by use of the TEF

560 margin legend.

561 THR **Threads**

562 The functionality described is optional. The functionality described is also an extension to the

563 ISO C standard.

564 Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.

565 Where additional semantics apply to a function, the material is identified by use of the THR

566		margin legend.
567	TMO	Timeouts
568		The functionality described is optional. The functionality described is also an extension to the
569		ISO C standard.
570		Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
571		Where additional semantics apply to a function, the material is identified by use of the TMO
572		margin legend.
573	TMR	Timers
574		The functionality described is optional. The functionality described is also an extension to the
575		ISO C standard.
576		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
577		Where additional semantics apply to a function, the material is identified by use of the TMR
578		margin legend.
579	TPI	Thread Priority Inheritance
580		The functionality described is optional. The functionality described is also an extension to the
581		ISO C standard.
582		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
583		Where additional semantics apply to a function, the material is identified by use of the TPI
584		margin legend.
585	TPP	Thread Priority Protection
586		The functionality described is optional. The functionality described is also an extension to the
587		ISO C standard.
588		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
589		Where additional semantics apply to a function, the material is identified by use of the TPP
590		margin legend.
591	TPS	Thread Execution Scheduling
592		The functionality described is optional. The functionality described is also an extension to the
593		ISO C standard.
594		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
595		Where additional semantics apply to a function, the material is identified by use of the TPS
596		margin legend.
597	TRC	Trace
598		The functionality described is optional. The functionality described is also an extension to the
599		ISO C standard.
600		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
601		Where additional semantics apply to a function, the material is identified by use of the TRC
602		margin legend.
603	TRI	Trace Inherit
604		The functionality described is optional. The functionality described is also an extension to the
605		ISO C standard.
606		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
607		Where additional semantics apply to a function, the material is identified by use of the TRI
608		margin legend.
609	TRL	Trace Log
610		The functionality described is optional. The functionality described is also an extension to the

611 ISO C standard.

612 Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.

613 Where additional semantics apply to a function, the material is identified by use of the TRL

614 margin legend.

615 TSA **Thread Stack Address Attribute**

616 The functionality described is optional. The functionality described is also an extension to the

617 ISO C standard.

618 Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.

619 Where additional semantics apply to a function, the material is identified by use of the TSA

620 margin legend.

621 TSF **Thread-Safe Functions**

622 The functionality described is optional. The functionality described is also an extension to the

623 ISO C standard.

624 Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.

625 Where additional semantics apply to a function, the material is identified by use of the TSF

626 margin legend.

627 TSH **Thread Process-Shared Synchronization**

628 The functionality described is optional. The functionality described is also an extension to the

629 ISO C standard.

630 Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.

631 Where additional semantics apply to a function, the material is identified by use of the TSH

632 margin legend.

633 TSP **Thread Sporadic Server**

634 The functionality described is optional. The functionality described is also an extension to the

635 ISO C standard.

636 Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.

637 Where additional semantics apply to a function, the material is identified by use of the TSP

638 margin legend.

639 TSS **Thread Stack Size Attribute**

640 The functionality described is optional. The functionality described is also an extension to the

641 ISO C standard.

642 Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.

643 Where additional semantics apply to a function, the material is identified by use of the TSS

644 margin legend.

645 TYM **Typed Memory Objects**

646 The functionality described is optional. The functionality described is also an extension to the

647 ISO C standard.

648 Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.

649 Where additional semantics apply to a function, the material is identified by use of the TYM

650 margin legend.

651 UP **User Portability Utilities**

652 The functionality described is optional.

653 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.

654 Where additional semantics apply to a utility, the material is identified by use of the UP margin

655 legend.

Extension

The functionality described is an XSI extension. Functionality marked XSI is also an extension to the ISO C standard. Application writers may confidently make use of an extension on all systems supporting the X/Open System Interfaces Extension.

If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that reference page is an extension. See the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.439, XSI.

XSI STREAMS

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the XSR margin legend.

1.9 Utility Limits

This section lists magnitude limitations imposed by a specific implementation. The braces notation, {LIMIT}, is used in this volume of IEEE Std 1003.1-2001 to indicate these values, but the braces are not part of the name.

Table 1-3 Utility Limit Minimum Values

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	2 048
{POSIX2_BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	1 000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order keyword in the locale definition file; see the border_start keyword in the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.2, <i>LC_COLLATE</i> .	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	2 048

Name	Description	Value
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$; see the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3.6, BREs Matching Multiple Characters.	255

The values specified in Table 1-3 (on page 17) represent the lowest values conforming implementations shall provide and, consequently, the largest values on which an application can rely without further enquiries, as described below. These values shall be accessible to applications via the *getconf* utility (see *getconf* (on page 484)).

Implementations may provide more liberal, or less restrictive, values than shown in Table 1-3 (on page 17). These possibly more liberal values are accessible using the symbols in Table 1-4.

The *sysconf()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 or the *getconf* utility return the value of each symbol on each specific implementation. The value so retrieved is the largest, or most liberal, value that is available throughout the session lifetime, as determined at session creation. The literal names shown in the table apply only to the *getconf* utility; the high-level language binding describes the exact form of each name to be used by the interfaces in that binding.

All numeric limits defined by the System Interfaces volume of IEEE Std 1003.1-2001, such as {PATH_MAX}, shall also apply to this volume of IEEE Std 1003.1-2001. All the utilities defined by this volume of IEEE Std 1003.1-2001 are implicitly limited by these values, unless otherwise noted in the utility descriptions.

It is not guaranteed that the application can actually reach the specified limit of an implementation in any given case, or at all, as a lack of virtual memory or other resources may prevent this. The limit value indicates only that the implementation does not specifically impose any arbitrary, more restrictive limit.

Table 1-4 Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order keyword in the locale definition file; see the	{POSIX2_COLL_WEIGHTS_MAX}

Name	Description	Minimum Value
{EXPR_NEST_MAX}	order_start keyword in the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.2, LC_COLLATE. The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	{POSIX2_EXPR_NEST_MAX}
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	{POSIX2_LINE_MAX}
{RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$; see the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3.6, BREs Matching Multiple Characters.	{POSIX2_RE_DUP_MAX}

The following value may be a constant within an implementation or may vary from one pathname to another.

{POSIX2_SYMLINKS}

When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2_SYMLINKS} is undefined.

1.10 Grammar Conventions

Portions of this volume of IEEE Std 1003.1-2001 are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input, suitable for program use; the lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial *yacc* code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

The following typographical conventions are used in the grammar; they have no significance except to aid in reading.

- The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar; for example, **While**, **Case**.)
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores; for example, **NEWLINE**, **ASSIGN_OP**, **NAME**.
- The identifiers for non-terminals are all lowercase.

1.11 Utility Description Defaults

This section describes all of the subsections used within the utility descriptions, including:

- Intended usage of the section
- Global defaults that affect all the standard utilities
- The meanings of notations used in this volume of IEEE Std 1003.1-2001 that are specific to individual utility sections

NAME

This section gives the name or names of the utility and briefly states its purpose.

SYNOPSIS

The SYNOPSIS section summarizes the syntax of the calling sequence for the utility, including options, option-arguments, and operands. Standards for utility naming are described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines; for describing the utility's arguments in the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.1, Utility Argument Syntax.

DESCRIPTION

The DESCRIPTION section describes the actions of the utility. If the utility has a very complex set of subcommands or its own procedural language, an EXTENDED DESCRIPTION section is also provided. Most explanations of optional functionality are omitted here, as they are usually explained in the OPTIONS section.

As stated in Section 1.7.1.11 (on page 7), some functions are described in terms of equivalent functionality. When specific functions are cited, the implementation shall provide equivalent functionality including side effects associated with successful execution of the function. The treatment of errors and intermediate results from the individual functions cited is generally not specified by this volume of IEEE Std 1003.1-2001. See the utility's EXIT STATUS and CONSEQUENCES OF ERRORS sections for all actions associated with errors encountered by the utility.

OPTIONS

The OPTIONS section describes the utility options and option-arguments, and how they modify the actions of the utility. Standard utilities that have options either fully comply with the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines or describe all deviations. Apparent disagreements between functionality descriptions in the OPTIONS and DESCRIPTION (or EXTENDED DESCRIPTION) sections are always resolved in favor of the OPTIONS section.

Each OPTIONS section that uses the phrase “The ... utility shall conform to the Utility Syntax Guidelines ...” refers only to the use of the utility as specified by this volume of IEEE Std 1003.1-2001; implementation extensions should also conform to the guidelines, but may allow exceptions for historical practice.

Unless otherwise stated in the utility description, when given an option unrecognized by the implementation, or when a required option-argument is not provided, standard utilities shall issue a diagnostic message to standard error and exit with a non-zero exit status.

All utilities in this volume of IEEE Std 1003.1-2001 shall be capable of processing arguments using eight-bit transparency.

Default Behavior: When this section is listed as “None.”, it means that the implementation need not support any options. Standard utilities that do not accept options, but that do accept operands, shall recognize “--” as a first argument to be discarded.

The requirement for recognizing “--” is because conforming applications need a way to shield their operands from any arbitrary options that the implementation may provide as an extension. For example, if the standard utility *foo* is listed as taking no options, and the application needed to give it a pathname with a leading hyphen, it could safely do it as:

```
foo -- -myfile
```

and avoid any problems with **-m** used as an extension.

OPERANDS

The OPERANDS section describes the utility operands, and how they affect the actions of the utility. Apparent disagreements between functionality descriptions in the OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be resolved in favor of the OPERANDS section.

If an operand naming a file can be specified as ‘-’, which means to use the standard input instead of a named file, this is explicitly stated in this section. Unless otherwise stated, the use of multiple instances of ‘-’ to mean standard input in a single command produces unspecified results.

Unless otherwise stated, the standard utilities that accept operands shall process those operands in the order specified in the command line.

Default Behavior: When this section is listed as “None.”, it means that the implementation need not support any operands.

STDIN

The STDIN section describes the standard input of the utility. This section is frequently merely a reference to the following section, as many utilities treat standard input and input files in the same manner. Unless otherwise stated, all restrictions described in the INPUT FILES section shall apply to this section as well.

Use of a terminal for standard input can cause any of the standard utilities that read standard input to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

The specified standard input format of the standard utilities shall not depend on the existence or value of the environment variables defined in this volume of IEEE Std 1003.1-2001, except as provided by this volume of IEEE Std 1003.1-2001.

Default Behavior: When this section is listed as “Not used.”, it means that the standard input shall not be read when the utility is used as described by this volume of IEEE Std 1003.1-2001.

INPUT FILES

The INPUT FILES section describes the files, other than the standard input, used as input by the utility. It includes files named as operands and option-arguments as well as other files that are referred to, such as start-up and initialization files, databases, and so on. Commonly-used files are generally described in one place and cross-referenced by other utilities.

All utilities in this volume of IEEE Std 1003.1-2001 shall be capable of processing input files using eight-bit transparency.

When a standard utility reads a seekable input file and terminates without an error before it reaches end-of-file, the utility shall ensure that the file offset in the open file description is properly positioned just past the last byte processed by the utility. For files that are not seekable, the state of the file offset in the open file description for that file is unspecified. A conforming application shall not assume that the following three commands are equivalent:

```
tail -n +2 file
(sed -n 1q; cat) < file
cat file | (sed -n 1q; cat)
```

The second command is equivalent to the first only when the file is seekable. The third command leaves the file offset in the open file description in an unspecified state. Other utilities, such as *head*, *read*, and *sh*, have similar properties.

Some of the standard utilities, such as filters, process input files a line or a block at a time and have no restrictions on the maximum input file size. Some utilities may have size limitations that are not as obvious as file space or memory limitations. Such limitations should reflect resource limitations of some sort, not arbitrary limits set by implementors. Implementations shall document those utilities that are limited by constraints other than file system space, available memory, and other limits specifically cited by this volume of IEEE Std 1003.1-2001, and identify what the constraint is and indicate a way of estimating when the constraint would be reached. Similarly, some utilities descend the directory tree (recursively). Implementations shall also document any limits that they may have in descending the directory tree that are beyond limits cited by this volume of IEEE Std 1003.1-2001.

When an input file is described as a “text file”, the utility produces undefined results if given input that is not from a text file, unless otherwise stated. Some utilities (for example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline> convention; unless otherwise stated, the utility need not be able to accumulate more than {LINE_MAX} bytes from a set of multiple, continued input lines. Thus, for a conforming application the total of all the continued lines in a set cannot exceed {LINE_MAX}. If a utility using the escaped <newline> convention detects an end-of-file condition immediately after an escaped <newline>, the results are unspecified.

Record formats are described in a notation similar to that used by the C-language function, *printf()*. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation for a description of this notation. The format description is intended to be sufficiently rigorous to allow other applications to generate these input files. However, since <blank>s can legitimately be included in some of the fields described by the standard utilities, particularly in locales other than the POSIX locale, this intent is not always realized.

Default Behavior: When this section is listed as “None.”, it means that no input files are required to be supplied when the utility is used as described by this volume of

IEEE Std 1003.1-2001.

ENVIRONMENT VARIABLES

The ENVIRONMENT VARIABLES section lists what variables affect the utility's execution.

The entire manner in which environment variables described in this volume of IEEE Std 1003.1-2001 affect the behavior of each utility is described in the ENVIRONMENT VARIABLES section for that utility, in conjunction with the global effects of the *LANG*, *LC_ALL*, and *NLSPATH* environment variables described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables. The existence or value of environment variables described in this volume of IEEE Std 1003.1-2001 shall not otherwise affect the specified behavior of the standard utilities. Any effects of the existence or value of environment variables not described by this volume of IEEE Std 1003.1-2001 upon the standard utilities are unspecified.

For those standard utilities that use environment variables as a means for selecting a utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to the path search described for *PATH* in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

All utilities in this volume of IEEE Std 1003.1-2001 shall be capable of processing environment variable names and values using eight-bit transparency.

Default Behavior: When this section is listed as “None.”, it means that the behavior of the utility is not directly affected by environment variables described by this volume of IEEE Std 1003.1-2001 when the utility is used as described by this volume of IEEE Std 1003.1-2001.

ASYNCHRONOUS EVENTS

The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as signals and what signals are caught.

Default Behavior: When this section is listed as “Default.”, or it refers to “the standard action for all other signals; see Section 1.11 (on page 20)” it means that the action taken as a result of the signal shall be one of the following:

1. The action shall be that inherited from the parent according to the rules of inheritance of signal actions defined in the System Interfaces volume of IEEE Std 1003.1-2001.
2. When no action has been taken to change the default, the default action shall be that specified by the System Interfaces volume of IEEE Std 1003.1-2001.
3. The result of the utility's execution is as if default actions had been taken.

A utility is permitted to catch a signal, perform some additional processing (such as deleting temporary files), restore the default signal action (or action inherited from the parent process), and resignal itself.

STDOUT

The STDOUT section completely describes the standard output of the utility. This section is frequently merely a reference to the following section, OUTPUT FILES, because many utilities treat standard output and output files in the same manner.

Use of a terminal for standard output may cause any of the standard utilities that write standard output to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

Record formats are described in a notation similar to that used by the C-language function, *printf()*. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation for a description of this notation.

The specified standard output of the standard utilities shall not depend on the existence or value of the environment variables defined in this volume of IEEE Std 1003.1-2001, except as provided by this volume of IEEE Std 1003.1-2001.

Some of the standard utilities describe their output using the verb *display*, defined in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.132, Display. Output described in the STDOUT sections of such utilities may be produced using means other than standard output. When standard output is directed to a terminal, the output described shall be written directly to the terminal. Otherwise, the results are undefined.

Default Behavior: When this section is listed as “Not used.”, it means that the standard output shall not be written when the utility is used as described by this volume of IEEE Std 1003.1-2001.

STDERR

The STDERR section describes the standard error output of the utility. Only those messages that are purposely sent by the utility are described.

Use of a terminal for standard error may cause any of the standard utilities that write standard error output to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

The format of diagnostic messages for most utilities is unspecified, but the language and cultural conventions of diagnostic and informative messages whose format is unspecified by this volume of IEEE Std 1003.1-2001 should be affected by the setting of *LC_MESSAGES* and *NLSPATH*.

The specified standard error output of standard utilities shall not depend on the existence or value of the environment variables defined in this volume of IEEE Std 1003.1-2001, except as provided by this volume of IEEE Std 1003.1-2001.

Default Behavior: When this section is listed as “The standard error shall be used only for diagnostic messages.”, it means that, unless otherwise stated, the diagnostic messages shall be sent to the standard error only when the exit status is non-zero and the utility is used as described by this volume of IEEE Std 1003.1-2001.

When this section is listed as “Not used.”, it means that the standard error shall not be used when the utility is used as described in this volume of IEEE Std 1003.1-2001.

OUTPUT FILES

The OUTPUT FILES section completely describes the files created or modified by the utility. Temporary or system files that are created for internal usage by this utility or other parts of the implementation (for example, spool, log, and audit files) are not described in this, or any, section. The utilities creating such files and the names of such files are unspecified. If applications are written to use temporary or intermediate files, they should use the *TMPDIR* environment variable, if it is set and represents an accessible directory, to select the location of temporary files.

Implementations shall ensure that temporary files, when used by the standard utilities, are named so that different utilities or multiple instances of the same utility can operate simultaneously without regard to their working directories, or any other process characteristic other than process ID. There are two exceptions to this rule:

1. Resources for temporary files other than the name space (for example, disk space, available directory entries, or number of processes allowed) are not guaranteed.
2. Certain standard utilities generate output files that are intended as input for other utilities (for example, *lex* generates **lex.yy.c**), and these cannot have unique names. These cases are explicitly identified in the descriptions of the respective utilities.

Any temporary file created by the implementation shall be removed by the implementation upon a utility's successful exit, exit because of errors, or before termination by any of the SIGHUP, SIGINT, or SIGTERM signals, unless specified otherwise by the utility description.

Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging mode) that would bypass any attempted recovery actions.

Record formats are described in a notation similar to that used by the C-language function, *printf()*; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation for a description of this notation.

Default Behavior: When this section is listed as “None.”, it means that no files are created or modified as a consequence of direct action on the part of the utility when the utility is used as described by this volume of IEEE Std 1003.1-2001. However, the utility may create or modify system files, such as log files, that are outside the utility's normal execution environment.

EXTENDED DESCRIPTION

The EXTENDED DESCRIPTION section provides a place for describing the actions of very complicated utilities, such as text editors or language processors, which typically have elaborate command languages.

Default Behavior: When this section is listed as “None.”, no further description is necessary.

EXIT STATUS

The EXIT STATUS section describes the values the utility shall return to the calling program, or shell, and the conditions that cause these values to be returned. Usually, utilities return zero for successful completion and values greater than zero for various error conditions. If specific numeric values are listed in this section, the system shall use those values for the errors described. In some cases, status values are listed more loosely, such as >0. A strictly conforming application shall not rely on any specific value in the range shown and shall be prepared to receive any value in the range.

For example, a utility may list zero as a successful return, 1 as a failure for a specific reason, and >1 as “an error occurred”. In this case, unspecified conditions may cause a 2 or 3, or other value, to be returned. A conforming application should be written so that it tests for successful exit status values (zero in this case), rather than relying upon the single specific error value listed in this volume of IEEE Std 1003.1-2001. In that way, it has maximum portability, even on implementations with extensions.

Unspecified error conditions may be represented by specific values not listed in this volume of IEEE Std 1003.1-2001.

CONSEQUENCES OF ERRORS

The CONSEQUENCES OF ERRORS section describes the effects on the environment, file systems, process state, and so on, when error conditions occur. It does not describe error messages produced or exit status values used.

The many reasons for failure of a utility are generally not specified by the utility descriptions. Utilities may terminate prematurely if they encounter: invalid usage of options, arguments, or environment variables; invalid usage of the complex syntaxes expressed in EXTENDED DESCRIPTION sections; difficulties accessing, creating, reading, or writing files; or difficulties associated with the privileges of the process.

The following shall apply to each utility, unless otherwise stated:

- If the requested action cannot be performed on an operand representing a file, directory, user, process, and so on, the utility shall issue a diagnostic message to standard error and continue processing the next operand in sequence, but the final exit status shall be returned as non-zero.

For a utility that recursively traverses a file hierarchy (such as *find* or *chown -R*), if the requested action cannot be performed on a file or directory encountered in the hierarchy, the utility shall issue a diagnostic message to standard error and continue processing the remaining files in the hierarchy, but the final exit status shall be returned as non-zero.

- If the requested action characterized by an option or option-argument cannot be performed, the utility shall issue a diagnostic message to standard error and the exit status returned shall be non-zero.
- When an unrecoverable error condition is encountered, the utility shall exit with a non-zero exit status.
- A diagnostic message shall be written to standard error whenever an error condition occurs.

When a utility encounters an error condition several actions are possible, depending on the severity of the error and the state of the utility. Included in the possible actions of various utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; validity checking of the file system or directory.

Default Behavior: When this section is listed as “Default.”, it means that any changes to the environment are unspecified.

APPLICATION USAGE

This section is informative.

The APPLICATION USAGE section gives advice to the application programmer or user about the way the utility should be used.

EXAMPLES

This section is informative.

The EXAMPLES section gives one or more examples of usage, where appropriate. In the event of conflict between an example and a normative part of the specification, the normative material is to be taken as correct.

In all examples, quoting has been used, showing how sample commands (utility names combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to the *system()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001. Such quoting would not be used if the utility is invoked using one of the *exec* functions defined in the System Interfaces volume of IEEE Std 1003.1-2001.

RATIONALE

This section is informative.

1105 This section contains historical information concerning the contents of this volume of
 1106 IEEE Std 1003.1-2001 and why features were included or discarded by the standard
 1107 developers.

1108 **FUTURE DIRECTIONS**

1109 This section is informative.

1110 The FUTURE DIRECTIONS section should be used as a guide to current thinking; there
 1111 is not necessarily a commitment to implement all of these future directions in their
 1112 entirety.

1113 **SEE ALSO**

1114 This section is informative.

1115 The SEE ALSO section lists related entries.

1116 **CHANGE HISTORY**

1117 This section is informative.

1118 This section shows the derivation of the entry and any significant changes that have
 1119 been made to it.

1120 Certain of the standard utilities describe how they can invoke other utilities or applications, such
 1121 as by passing a command string to the command interpreter. The external influences (STDIN,
 1122 ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF
 1123 ERRORS, and so on) of such invoked utilities are not described in the section concerning the
 1124 standard utility that invokes them.

1125 **1.12 Considerations for Utilities in Support of Files of Arbitrary Size**

1126 The following utilities support files of any size up to the maximum that can be created by the
 1127 implementation. This support includes correct writing of file size-related values (such as file
 1128 sizes and offsets, line numbers, and block counts) and correct interpretation of command line
 1129 arguments that contain such values.

1130 *basename* Return non-directory portion of pathname.

1131 *cat* Concatenate and print files.

1132 *cd* Change working directory.

1133 *chgrp* Change file group ownership.

1134 *chmod* Change file modes.

1135 *chown* Change file ownership.

1136 *cksum* Write file checksums and sizes.

1137 *cmp* Compare two files.

1138 *cp* Copy files.

1139 *dd* Convert and copy a file.

1140 *df* Report free disk space.

1141 *dirname* Return directory portion of pathname.

1142 *du* Estimate file space usage.

1143	<i>find</i>	Find files.
1144	<i>ln</i>	Link files.
1145	<i>ls</i>	List directory contents.
1146	<i>mkdir</i>	Make directories.
1147	<i>mv</i>	Move files.
1148	<i>pathchk</i>	Check pathnames.
1149	<i>pwd</i>	Return working directory name.
1150	<i>rm</i>	Remove directory entries.
1151	<i>rmdir</i>	Remove directories.
1152	<i>sh</i>	Shell, the standard command language interpreter.
1153	<i>sum</i>	Print checksum and block or byte count of a file.
1154	<i>test</i>	Evaluate expression.
1155	<i>touch</i>	Change file access and modification times.
1156	<i>ulimit</i>	Set or report file size limit.
1157	Exceptions to the requirement that utilities support files of any size up to the maximum are as follows:	
1158		
1159	1. Uses of files as command scripts, or for configuration or control, are exempt. For example,	
1160	it is not required that <i>sh</i> be able to read an arbitrarily large .profile .	
1161	2. Shell input and output redirection are exempt. For example, it is not required that the	
1162	redirections <i>sum</i> < <i>file</i> or <i>echo foo</i> > <i>file</i> succeed for an arbitrarily large existing file.	

1163 1.13 Built-In Utilities

1164 Any of the standard utilities may be implemented as regular built-in utilities within the
 1165 command language interpreter. This is usually done to increase the performance of frequently
 1166 used utilities or to achieve functionality that would be more difficult in a separate environment.
 1167 The utilities named in Table 1-5 are frequently provided in built-in form. All of the utilities
 1168 named in the table have special properties in terms of command search order within the shell, as
 1169 described in Section 2.9.1.1 (on page 48).

1170 **Table 1-5** Regular Built-In Utilities

1171	<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
1172	<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>true</i>	
1173	<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>umask</i>	
1174	<i>command</i>	<i>getopts</i>	<i>pwd</i>	<i>unalias</i>	

1175 However, all of the standard utilities, including the regular built-ins in the table, but not the
 1176 special built-ins described in Section 2.14 (on page 64), shall be implemented in a manner so that
 1177 they can be accessed via the *exec* family of functions as defined in the System Interfaces volume
 1178 of IEEE Std 1003.1-2001 and can be invoked directly by those standard utilities that require it |
 1179 (*env*, *find*, *nice*, *nohup*, *time*, *xargs*).

Shell Command Language

1180

1181 This chapter contains the definition of the Shell Command Language.

1182 2.1 Shell Introduction

1183 The shell is a command language interpreter. This chapter describes the syntax of that command
 1184 language as it is used by the *sh* utility and the *system()* and *popen()* functions defined in the
 1185 System Interfaces volume of IEEE Std 1003.1-2001.

1186 The shell operates according to the following general overview of operations. The specific
 1187 details are included in the cited sections of this chapter.

- 1188 1. The shell reads its input from a file (see *sh*), from the `-c` option or from the *system()* and
 1189 *popen()* functions defined in the System Interfaces volume of IEEE Std 1003.1-2001. If the
 1190 first line of a file of shell commands starts with the characters "#!", the results are
 1191 unspecified.
- 1192 2. The shell breaks the input into tokens: words and operators; see Section 2.3 (on page 31).
- 1193 3. The shell parses the input into simple commands (see Section 2.9.1 (on page 47)) and
 1194 compound commands (see Section 2.9.4 (on page 52)).
- 1195 4. The shell performs various expansions (separately) on different parts of each command,
 1196 resulting in a list of pathnames and fields to be treated as a command and arguments; see
 1197 Section 2.6 (on page 36).
- 1198 5. The shell performs redirection (see Section 2.7 (on page 43)) and removes redirection
 1199 operators and their operands from the parameter list.
- 1200 6. The shell executes a function (see Section 2.9.5 (on page 54)), built-in (see Section 2.14 (on
 1201 page 64)), executable file, or script, giving the names of the arguments as positional
 1202 parameters numbered 1 to *n*, and the name of the command (or in the case of a function
 1203 within a script, the name of the script) as the positional parameter numbered 0 (see Section
 1204 2.9.1.1 (on page 48)).
- 1205 7. The shell optionally waits for the command to complete and collects the exit status (see
 1206 Section 2.8.2 (on page 46)).

2.2 Quoting

Quoting is used to remove the special meaning of certain characters or words to the shell. Quoting can be used to preserve the literal meaning of the special characters in the next paragraph, prevent reserved words from being recognized as such, and prevent parameter expansion and command substitution within here-document processing (see Section 2.7.4 (on page 44)).

The application shall quote the following characters if they are to represent themselves:

| & ; < > () \$ ' \ " ' <space> <tab> <newline>

and the following may need to be quoted under certain circumstances. That is, these characters may be special depending on conditions described elsewhere in this volume of IEEE Std 1003.1-2001:

* ? [# ~ = %

The various quoting mechanisms are the escape character, single-quotes, and double-quotes. The here-document represents another form of quoting; see Section 2.7.4 (on page 44).

2.2.1 Escape Character (Backslash)

A backslash that is not quoted shall preserve the literal value of the following character, with the exception of a <newline>. If a <newline> follows the backslash, the shell shall interpret this as line continuation. The backslash and <newline>s shall be removed before splitting the input into tokens. Since the escaped <newline> is removed entirely from the input and is not replaced by any white space, it cannot serve as a token separator.

2.2.2 Single-Quotes

Enclosing characters in single-quotes (' ') shall preserve the literal value of each character within the single-quotes. A single-quote cannot occur within single-quotes.

2.2.3 Double-Quotes

Enclosing characters in double-quotes (" ") shall preserve the literal value of all characters within the double-quotes, with the exception of the characters dollar sign, backquote, and backslash, as follows:

\$ The dollar sign shall retain its special meaning introducing parameter expansion (see Section 2.6.2 (on page 37)), a form of command substitution (see Section 2.6.3 (on page 40)), and arithmetic expansion (see Section 2.6.4 (on page 41)).

The input characters within the quoted string that are also enclosed between "\$ (" and the matching ') ' shall not be affected by the double-quotes, but rather shall define that command whose output replaces the "\$ (. . .) " when the word is expanded. The tokenizing rules in Section 2.3 (on page 31), not including the alias substitutions in Section 2.3.1 (on page 32), shall be applied recursively to find the matching ') ' .

Within the string of characters from an enclosed "\$ { " to the matching ' } ' , an even number of unescaped double-quotes or single-quotes, if any, shall occur. A preceding backslash character shall be used to escape a literal ' { ' or ' } ' . The rule in Section 2.6.2 (on page 37) shall be used to determine the matching ' } ' .

` The backquote shall retain its special meaning introducing the other form of command substitution (see Section 2.6.3 (on page 40)). The portion of the quoted string from the initial backquote and the characters up to the next backquote that is not preceded by a backslash,

1249 having escape characters removed, defines that command whose output replaces "`\ . . \`"
 1250 when the word is expanded. Either of the following cases produces undefined results:

- 1251 • A single-quoted or double-quoted string that begins, but does not end, within the
 1252 "`\ . . \`" sequence
- 1253 • A "`\ . . \`" sequence that begins, but does not end, within the same double-quoted
 1254 string

1255 \ The backslash shall retain its special meaning as an escape character (see Section 2.2.1 (on
 1256 page 30)) only when followed by one of the following characters when considered special:

1257 \$ \ " \ <newline>

1258 The application shall ensure that a double-quote is preceded by a backslash to be included
 1259 within double-quotes. The parameter '@' has special meaning inside double-quotes and is
 1260 described in Section 2.5.2 (on page 34).

1261 2.3 Token Recognition

1262 The shell shall read its input in terms of lines from a file, from a terminal in the case of an
 1263 interactive shell, or from a string in the case of `sh -c` or `system()`. The input lines can be of
 1264 unlimited length. These lines shall be parsed using two major modes: ordinary token recognition
 1265 and processing of here-documents.

1266 When an **io_here** token has been recognized by the grammar (see Section 2.10 (on page 55)), one
 1267 or more of the subsequent lines immediately following the next **NEWLINE** token form the body
 1268 of one or more here-documents and shall be parsed according to the rules of Section 2.7.4 (on
 1269 page 44).

1270 When it is not processing an **io_here**, the shell shall break its input into tokens by applying the
 1271 first applicable rule below to the next character in its input. The token shall be from the current
 1272 position in the input until a token is delimited according to one of the rules below; the characters
 1273 forming the token are exactly those in the input, including any quoting characters. If it is
 1274 indicated that a token is delimited, and no characters have been included in a token, processing
 1275 shall continue until an actual token is delimited.

- 1276 1. If the end of input is recognized, the current token shall be delimited. If there is no current
 1277 token, the end-of-input indicator shall be returned as the token.
- 1278 2. If the previous character was used as part of an operator and the current character is not
 1279 quoted and can be used with the current characters to form an operator, it shall be used as
 1280 part of that (operator) token.
- 1281 3. If the previous character was used as part of an operator and the current character cannot
 1282 be used with the current characters to form an operator, the operator containing the
 1283 previous character shall be delimited.
- 1284 4. If the current character is backslash, single-quote, or double-quote (`'\'`, `'\''`, or `'\"'`) and
 1285 it is not quoted, it shall affect quoting for subsequent characters up to the end of the quoted
 1286 text. The rules for quoting are as described in Section 2.2 (on page 30). During token
 1287 recognition no substitutions shall be actually performed, and the result token shall contain
 1288 exactly the characters that appear in the input (except for `<newline>` joining), unmodified,
 1289 including any embedded or enclosing quotes or substitution operators, between the quote
 1290 mark and the end of the quoted text. The token shall not be delimited by the end of the
 1291 quoted field.

5. If the current character is an unquoted '\$' or '\', the shell shall identify the start of any candidates for parameter expansion (Section 2.6.2 (on page 37)), command substitution (Section 2.6.3 (on page 40)), or arithmetic expansion (Section 2.6.4 (on page 41)) from their introductory unquoted character sequences: '\$' or "\${", "\$(" or '\', and "\$(", respectively. The shell shall read sufficient input to determine the end of the unit to be expanded (as explained in the cited sections). While processing the characters, if instances of expansions or quoting are found nested within the substitution, the shell shall recursively process them in the manner specified for the construct that is found. The characters found from the beginning of the substitution to its end, allowing for any recursion necessary to recognize embedded constructs, shall be included unmodified in the result token, including any embedded or enclosing substitution operators or quotes. The token shall not be delimited by the end of the substitution.
6. If the current character is not quoted and can be used as the first character of a new operator, the current token (if any) shall be delimited. The current character shall be used as the beginning of the next (operator) token.
7. If the current character is an unquoted <newline>, the current token shall be delimited.
8. If the current character is an unquoted <blank>, any token containing the previous character is delimited and the current character shall be discarded.
9. If the previous character was part of a word, the current character shall be appended to that word.
10. If the current character is a '#', it and all subsequent characters up to, but excluding, the next <newline> shall be discarded as a comment. The <newline> that ends the line is not considered part of the comment.
11. The current character is used as the start of a new word.

Once a token is delimited, it is categorized as required by the grammar in Section 2.10 (on page 55).

2.3.1 Alias Substitution

UP XSI The processing of aliases shall be supported on all XSI-conformant systems or if the system supports the User Portability Utilities option (and the rest of this section is not further shaded for these options).

After a token has been delimited, but before applying the grammatical rules in Section 2.10 (on page 55), a resulting word that is identified to be the command name word of a simple command shall be examined to determine whether it is an unquoted, valid alias name. However, reserved words in correct grammatical context shall not be candidates for alias substitution. A valid alias name (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.10, Alias Name) shall be one that has been defined by the *alias* utility and not subsequently undefined using *unalias*. Implementations also may provide predefined valid aliases that are in effect when the shell is invoked. To prevent infinite loops in recursive aliasing, if the shell is not currently processing an alias of the same name, the word shall be replaced by the value of the alias; otherwise, it shall not be replaced.

If the value of the alias replacing the word ends in a <blank>, the shell shall check the next command word for alias substitution; this process shall continue until a word is found that is not a valid alias or an alias value does not end in a <blank>.

When used as specified by this volume of IEEE Std 1003.1-2001, alias definitions shall not be inherited by separate invocations of the shell or by the utility execution environments invoked by the shell; see Section 2.12 (on page 61).

2.4 Reserved Words

Reserved words are words that have special meaning to the shell; see Section 2.9 (on page 47). The following words shall be recognized as reserved words:

!	do	esac	in
{	done	fi	then
}	elif	for	until
case	else	if	while

This recognition shall only occur when none of the characters is quoted and when the word is used as:

- The first word of a command
- The first word following one of the reserved words other than **case**, **for**, or **in**
- The third word in a **case** command (only **in** is valid in this case)
- The third word in a **for** command (only **in** and **do** are valid in this case)

See the grammar in Section 2.10 (on page 55).

The following words may be recognized as reserved words on some implementations (when none of the characters are quoted), causing unspecified results:

[[	]]	function	select

Words that are the concatenation of a name and a colon (':') are reserved; their use produces unspecified results.

2.5 Parameters and Variables

A parameter can be denoted by a name, a number, or one of the special characters listed in Section 2.5.2 (on page 34). A variable is a parameter denoted by a name.

A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can only be unset by using the *unset* special built-in command.

2.5.1 Positional Parameters

A positional parameter is a parameter denoted by the decimal value represented by one or more digits, other than the single digit 0. The digits denoting the positional parameters shall always be interpreted as a decimal value, even if there is a leading zero. When a positional parameter with more than one digit is specified, the application shall enclose the digits in braces (see Section 2.6.2 (on page 37)). Positional parameters are initially assigned when the shell is invoked (see *sh*), temporarily replaced when a shell function is invoked (see Section 2.9.5 (on page 54)), and can be reassigned with the *set* special built-in command.

2.5.2 Special Parameters

Listed below are the special parameters and the values to which they shall expand. Only the values of the special parameters are listed; see Section 2.6 (on page 36) for a detailed summary of all the stages involved in expanding words.

@ Expands to the positional parameters, starting from one. When the expansion occurs within double-quotes, and where field splitting (see Section 2.6.5 (on page 42)) is performed, each positional parameter shall expand as a separate field, with the provision that the expansion of the first parameter shall still be joined with the beginning part of the original word (assuming that the expanded parameter was embedded within a word), and the expansion of the last parameter shall still be joined with the last part of the original word. If there are no positional parameters, the expansion of '@' shall generate zero fields, even when '@' is double-quoted.

***** Expands to the positional parameters, starting from one. When the expansion occurs within a double-quoted string (see Section 2.2.3 (on page 30)), it shall expand to a single field with the value of each parameter separated by the first character of the *IFS* variable, or by a <space> if *IFS* is unset. If *IFS* is set to a null string, this is not equivalent to unsetting it; its first character does not exist, so the parameter values are concatenated.

Expands to the decimal number of positional parameters. The command name (parameter 0) shall not be counted in the number given by '#' because it is a special parameter, not a positional parameter.

? Expands to the decimal exit status of the most recent pipeline (see Section 2.9.2 (on page 49)).

– (Hyphen.) Expands to the current option flags (the single-letter option names concatenated into a string) as specified on invocation, by the *set* special built-in command, or implicitly by the shell.

\$ Expands to the decimal process ID of the invoked shell. In a subshell (see Section 2.12 (on page 61)), '\$' shall expand to the same value as that of the current shell.

! Expands to the decimal process ID of the most recent background command (see Section 2.9.3 (on page 50)) executed from the current shell. (For example, background commands executed from subshells do not affect the value of "\$!" in the current shell environment.) For a pipeline, the process ID is that of the last command in the pipeline.

0 (Zero.) Expands to the name of the shell or shell script. See *sh* (on page 850) for a detailed description of how this name is derived.

See the description of the *IFS* variable in Section 2.5.3.

2.5.3 Shell Variables

Variables shall be initialized from the environment (as defined by the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables and the *exec* function in the System Interfaces volume of IEEE Std 1003.1-2001) and can be given new values with variable assignment commands. If a variable is initialized from the environment, it shall be marked for export immediately; see the *export* special built-in. New variables can be defined and initialized with variable assignments, with the *read* or *getopts* utilities, with the *name* parameter in a **for** loop, with the $\${name=word}$ expansion, or with other mechanisms provided as implementation extensions.

The following variables shall affect the execution of the shell:

1414	UP XSI	ENV	The processing of the <i>ENV</i> shell variable shall be supported on all XSI-conformant systems or if the system supports the User Portability Utilities option.
1415			
1416			
1417			This variable, when and only when an interactive shell is invoked, shall be
1418			subjected to parameter expansion (see Section 2.6.2 (on page 37)) by the shell
1419			and the resulting value shall be used as a pathname of a file containing shell
1420			commands to execute in the current environment. The file need not be
1421			executable. If the expanded value of <i>ENV</i> is not an absolute pathname, the
1422			results are unspecified. <i>ENV</i> shall be ignored if the user's real and effective
1423			user IDs or real and effective group IDs are different.
1424		HOME	The pathname of the user's home directory. The contents of <i>HOME</i> are used in
1425			tilde expansion (see Section 2.6.1 (on page 37)).
1426		IFS	(Input Field Separators.) A string treated as a list of characters that is used for
1427			field splitting and to split lines into fields with the <i>read</i> command. If <i>IFS</i> is not
1428			set, the shell shall behave as if the value of <i>IFS</i> is <space>, <tab>, and
1429			<newline>; see Section 2.6.5 (on page 42). Implementations may ignore the
1430			value of <i>IFS</i> in the environment at the time the shell is invoked, treating <i>IFS</i> as
1431			if it were not set.
1432		LANG	Provide a default value for the internationalization variables that are unset or
1433			null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,
1434			Internationalization Variables for the precedence of internationalization
1435			variables used to determine the values of locale categories.)
1436		LC_ALL	The value of this variable overrides the <i>LC_*</i> variables and <i>LANG</i> , as described
1437			in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8,
1438			Environment Variables.
1439		LC_COLLATE	Determine the behavior of range expressions, equivalence classes, and multi-
1440			character collating elements within pattern matching.
1441		LC_CTYPE	Determine the interpretation of sequences of bytes of text data as characters
1442			(for example, single-byte as opposed to multi-byte characters), which
1443			characters are defined as letters (character class alpha) and <blank>s
1444			(character class blank), and the behavior of character classes within pattern
1445			matching. Changing the value of <i>LC_CTYPE</i> after the shell has started shall
1446			not affect the lexical processing of shell commands in the current shell
1447			execution environment or its subshells. Invoking a shell script or performing
1448			<i>exec sh</i> subjects the new shell to the changes in <i>LC_CTYPE</i> .
1449		LC_MESSAGES	Determine the language in which messages should be written.
1450		LINENO	Set by the shell to a decimal number representing the current sequential line
1451			number (numbered starting with 1) within a script or function before it
1452			executes each command. If the user unsets or resets <i>LINENO</i> , the variable may
1453			lose its special meaning for the life of the shell. If the shell is not currently
1454			executing a script or function, the value of <i>LINENO</i> is unspecified. This
1455			volume of IEEE Std 1003.1-2001 specifies the effects of the variable only for
1456			systems supporting the User Portability Utilities option.
1457	XSI	NLSPATH	Determine the location of message catalogs for the processing of
1458			<i>LC_MESSAGES</i> .
1459		PATH	A string formatted as described in the Base Definitions volume of
1460			IEEE Std 1003.1-2001, Chapter 8, Environment Variables, used to effect

1461		command interpretation; see Section 2.9.1.1 (on page 48).
1462	<i>PPID</i>	Set by the shell to the decimal process ID of the process that invoked this shell.
1463		In a subshell (see Section 2.12 (on page 61)), <i>PPID</i> shall be set to the same
1464		value as that of the parent of the current shell. For example, <i>echo \$PPID</i> and
1465		(<i>echo \$PPID</i>) would produce the same value. This volume of
1466		IEEE Std 1003.1-2001 specifies the effects of the variable only for systems
1467		supporting the User Portability Utilities option.
1468	<i>PS1</i>	Each time an interactive shell is ready to read a command, the value of this
1469		variable shall be subjected to parameter expansion and written to standard
1470		error. The default value shall be "\$ ". For users who have specific additional
1471		implementation-defined privileges, the default may be another,
1472		implementation-defined value. The shell shall replace each instance of the
1473		character '!' in <i>PS1</i> with the history file number of the next command to be
1474		typed. Escaping the '!' with another '!' (that is, "!!") shall place the literal
1475		character '!' in the prompt. This volume of IEEE Std 1003.1-2001 specifies
1476		the effects of the variable only for systems supporting the User Portability
1477		Utilities option.
1478	<i>PS2</i>	Each time the user enters a <newline> prior to completing a command line in
1479		an interactive shell, the value of this variable shall be subjected to parameter
1480		expansion and written to standard error. The default value is "> ". This
1481		volume of IEEE Std 1003.1-2001 specifies the effects of the variable only for
1482		systems supporting the User Portability Utilities option.
1483	<i>PS4</i>	When an execution trace (<i>set -x</i>) is being performed in an interactive shell,
1484		before each line in the execution trace, the value of this variable shall be
1485		subjected to parameter expansion and written to standard error. The default
1486		value is "+ ". This volume of IEEE Std 1003.1-2001 specifies the effects of the
1487		variable only for systems supporting the User Portability Utilities option.
1488	<i>PWD</i>	Set by the shell to be an absolute pathname of the current working directory,
1489		containing no components of type symbolic link, no components that are dot,
1490		and no components that are dot-dot when the shell is initialized. If an
1491		application sets or unsets the value of <i>PWD</i> , the behaviors of the <i>cd</i> and <i>pwd</i>
1492		utilities are unspecified.

1493 2.6 Word Expansions

1494 This section describes the various expansions that are performed on words. Not all expansions
1495 are performed on every word, as explained in the following sections.

1496 Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and
1497 quote removals that occur within a single word expand to a single field. It is only field splitting
1498 or pathname expansion that can create multiple fields from a single word. The single exception
1499 to this rule is the expansion of the special parameter '@' within double-quotes, as described in
1500 Section 2.5.2 (on page 34).

1501 The order of word expansion shall be as follows:

- 1502 1. Tilde expansion (see Section 2.6.1 (on page 37)), parameter expansion (see Section 2.6.2 (on
1503 page 37)), command substitution (see Section 2.6.3 (on page 40)), and arithmetic expansion
1504 (see Section 2.6.4 (on page 41)) shall be performed, beginning to end. See item 5 in Section
1505 2.3 (on page 31).

2. Field splitting (see Section 2.6.5 (on page 42)) shall be performed on the portions of the fields generated by step 1, unless *IFS* is null.
3. Pathname expansion (see Section 2.6.6 (on page 42)) shall be performed, unless *set -f* is in effect.
4. Quote removal (see Section 2.6.7 (on page 42)) shall always be performed last.

The expansions described in this section shall occur in the same shell environment as that in which the command is executed.

If the complete expansion appropriate for a word results in an empty field, that empty field shall be deleted from the list of fields that form the completely expanded command, unless the original word contained single-quote or double-quote characters.

The '\$' character is used to introduce parameter expansion, command substitution, or arithmetic evaluation. If an unquoted '\$' is followed by a character that is either not numeric, the name of one of the special parameters (see Section 2.5.2 (on page 34)), a valid first character of a variable name, a left curly brace ('{') or a left parenthesis, the result is unspecified.

2.6.1 Tilde Expansion

A “tilde-prefix” consists of an unquoted tilde character at the beginning of a word, followed by all of the characters preceding the first unquoted slash in the word, or all the characters in the word if there is no slash. In an assignment (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.21, Variable Assignment), multiple tilde-prefixes can be used: at the beginning of the word (that is, following the equal sign of the assignment), following any unquoted colon, or both. A tilde-prefix in an assignment is terminated by the first unquoted colon or slash. If none of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the tilde are treated as a possible login name from the user database. A portable login name cannot contain characters outside the set given in the description of the *LOGNAME* environment variable in the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.3, Other Environment Variables. If the login name is null (that is, the tilde-prefix contains only the tilde), the tilde-prefix is replaced by the value of the variable *HOME*. If *HOME* is unset, the results are unspecified. Otherwise, the tilde-prefix shall be replaced by a pathname of the initial working directory associated with the login name obtained using the *getpwnam()* function as defined in the System Interfaces volume of IEEE Std 1003.1-2001. If the system does not recognize the login name, the results are undefined.

2.6.2 Parameter Expansion

The format for parameter expansion is as follows:

```
${expression}
```

where *expression* consists of all characters until the matching '}'. Any '}' escaped by a backslash or within a quoted string, and characters in embedded arithmetic expansions, command substitutions, and variable expansions, shall not be examined in determining the matching '}'.

The simplest form for parameter expansion is:

```
${parameter}
```

The value, if any, of *parameter* shall be substituted.

The parameter name or symbol can be enclosed in braces, which are optional except for positional parameters with more than one digit or when *parameter* is followed by a character that could be interpreted as part of the name. The matching closing brace shall be determined by

counting brace levels, skipping over enclosed quoted strings, and command substitutions.

If the parameter name or symbol is not enclosed in braces, the expansion shall use the longest valid name (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.230, Name), whether or not the symbol represented by that name exists.

If a parameter expansion occurs inside double-quotes:

- Pathname expansion shall not be performed on the results of the expansion.
- Field splitting shall not be performed on the results of the expansion, with the exception of '@'; see Section 2.5.2 (on page 34).

In addition, a parameter expansion can be modified by using one of the following formats. In each case that a value of *word* is needed (based on the state of *parameter*, as described below), *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and arithmetic expansion. If *word* is not needed, it shall not be expanded. The '}' character that delimits the following parameter expansion modifications shall be determined as described previously in this section and in Section 2.2.3 (on page 30). (For example, `${foo-bar}xyz` would result in the expansion of **foo** followed by the string **xyz** if **foo** is set, else the string "barxyz").

`${parameter:-word}` **Use Default Values.** If *parameter* is unset or null, the expansion of *word* shall be substituted; otherwise, the value of *parameter* shall be substituted.

`${parameter:=word}` **Assign Default Values.** If *parameter* is unset or null, the expansion of *word* shall be assigned to *parameter*. In all cases, the final value of *parameter* shall be substituted. Only variables, not positional parameters or special parameters, can be assigned in this way.

`${parameter:?[word]}` **Indicate Error if Null or Unset.** If *parameter* is unset or null, the expansion of *word* (or a message indicating it is unset if *word* is omitted) shall be written to standard error and the shell exits with a non-zero exit status. Otherwise, the value of *parameter* shall be substituted. An interactive shell need not exit.

`${parameter:+word}` **Use Alternative Value.** If *parameter* is unset or null, null shall be substituted; otherwise, the expansion of *word* shall be substituted.

In the parameter expansions shown previously, use of the colon in the format shall result in a test for a parameter that is unset or null; omission of the colon shall result in a test for a parameter that is only unset. The following table summarizes the effect of the colon:

	<i>parameter</i> Set and Not Null	<i>parameter</i> Set But Null	<i>parameter</i> Unset
<code>\${parameter:-word}</code>	substitute <i>parameter</i>	substitute <i>word</i>	substitute <i>word</i>
<code>\${parameter-word}</code>	substitute <i>parameter</i>	substitute null	substitute <i>word</i>
<code>\${parameter:=word}</code>	substitute <i>parameter</i>	assign <i>word</i>	assign <i>word</i>
<code>\${parameter=word}</code>	substitute <i>parameter</i>	substitute null	assign <i>word</i>
<code>\${parameter:?word}</code>	substitute <i>parameter</i>	error, exit	error, exit
<code>\${parameter?word}</code>	substitute <i>parameter</i>	substitute null	error, exit
<code>\${parameter:+word}</code>	substitute <i>word</i>	substitute null	substitute null
<code>\${parameter+word}</code>	substitute <i>word</i>	substitute <i>word</i>	substitute null

In all cases shown with “substitute”, the expression is replaced with the value shown. In all cases shown with “assign”, *parameter* is assigned that value, which also replaces the expression.

1594 `${#parameter}` **String Length.** The length in characters of the value of *parameter* shall be
 1595 substituted. If *parameter* is `'*'` or `'@'`, the result of the expansion is
 1596 unspecified.

1597 The following four varieties of parameter expansion provide for substring processing. In each
 1598 case, pattern matching notation (see Section 2.13 (on page 62)), rather than regular expression
 1599 notation, shall be used to evaluate the patterns. If *parameter* is `'*'` or `'@'`, the result of the
 1600 expansion is unspecified. Enclosing the full parameter expansion string in double-quotes shall
 1601 not cause the following four varieties of pattern characters to be quoted, whereas quoting
 1602 characters within the braces shall have this effect.

1603 `${parameter%word}` **Remove Smallest Suffix Pattern.** The *word* shall be expanded to produce
 1604 a pattern. The parameter expansion shall then result in *parameter*, with the
 1605 smallest portion of the suffix matched by the *pattern* deleted.

1606 `${parameter%%word}` **Remove Largest Suffix Pattern.** The *word* shall be expanded to produce a
 1607 pattern. The parameter expansion shall then result in *parameter*, with the
 1608 largest portion of the suffix matched by the *pattern* deleted.

1609 `${parameter#word}` **Remove Smallest Prefix Pattern.** The *word* shall be expanded to produce
 1610 a pattern. The parameter expansion shall then result in *parameter*, with the
 1611 smallest portion of the prefix matched by the *pattern* deleted.

1612 `${parameter##word}` **Remove Largest Prefix Pattern.** The *word* shall be expanded to produce a
 1613 pattern. The parameter expansion shall then result in *parameter*, with the
 1614 largest portion of the prefix matched by the *pattern* deleted.

1615 Examples

1616 `${parameter:=word}`
 1617 In this example, *ls* is executed only if *x* is null or unset. (The `$(ls)` command substitution
 1618 notation is explained in Section 2.6.3 (on page 40).)

1619 `$ {x:-$(ls) }`

1620 `${parameter:=word}`
 1621 `unset X`
 1622 `echo ${X:=abc}`
 1623 `abc`

1624 `${parameter:?word}`
 1625 `unset posix`
 1626 `echo ${posix:?}`
 1627 `sh: posix: parameter null or not set`

1628 `${parameter:+word}`
 1629 `set a b c`
 1630 `echo ${3:+posix}`
 1631 `posix`

1632 `${#parameter}`
 1633 `HOME=/usr/posix`
 1634 `echo ${#HOME}`
 1635 `10`

1636 `${parameter%word}`
 1637 `x=file.c`
 1638 `echo ${x%.c}.o`

```

1639         file.o
1640     ${parameter%%word}
1641         x=posix/src/std
1642         echo ${x%*/}
1643         posix
1644     ${parameter#word}
1645         x=$HOME/src/cmd
1646         echo ${x#$HOME}
1647         /src/cmd
1648     ${parameter##word}
1649         x=/one/two/three
1650         echo ${x##*/}
1651         three

```

The double-quoting of patterns is different depending on where the double-quotes are placed:

```

1653 "$ {x#*} "    The asterisk is a pattern character.
1654 ${x#"*"}     The literal asterisk is quoted and not special.

```

2.6.3 Command Substitution

Command substitution allows the output of a command to be substituted in place of the command name itself. Command substitution shall occur when the command is enclosed as follows:

```
$ ( command )
```

or (backquoted version):

```
` command `
```

The shell shall expand the command substitution by executing *command* in a subshell environment (see Section 2.12 (on page 61)) and replacing the command substitution (the text of *command* plus the enclosing "\$ ()" or backquotes) with the standard output of the command, removing sequences of one or more <newline>s at the end of the substitution. Embedded <newline>s before the end of the output shall not be removed; however, they may be treated as field delimiters and eliminated during field splitting, depending on the value of *IFS* and quoting that is in effect.

Within the backquoted style of command substitution, backslash shall retain its literal meaning, except when followed by: '\$', '`', or '\`' (dollar sign, backquote, backslash). The search for the matching backquote shall be satisfied by the first backquote found without a preceding backslash; during this search, if a non-escaped backquote is encountered within a shell comment, a here-document, an embedded command substitution of the \$(*command*) form, or a quoted string, undefined results occur. A single-quoted or double-quoted string that begins, but does not end, within the "`...`" sequence produces undefined results.

With the \$(*command*) form, all characters following the open parenthesis to the matching closing parenthesis constitute the *command*. Any valid shell script can be used for *command*, except a script consisting solely of redirections which produces unspecified results.

The results of command substitution shall not be processed for further tilde expansion, parameter expansion, command substitution, or arithmetic expansion. If a command substitution occurs inside double-quotes, field splitting and pathname expansion shall not be performed on the results of the substitution.

Command substitution can be nested. To specify nesting within the backquoted version, the application shall precede the inner backquotes with backslashes, for example:

```
\`command\`
```

If the command substitution consists of a single subshell, such as:

```
$( (command) )
```

a conforming application shall separate the "\$(" and "(" into two tokens (that is, separate them with white space). This is required to avoid any ambiguities with arithmetic expansion.

2.6.4 Arithmetic Expansion

Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and substituting its value. The format for arithmetic expansion shall be as follows:

```
$( (expression) )
```

The expression shall be treated as if it were in double-quotes, except that a double-quote inside the expression is not treated specially. The shell shall expand all tokens in the expression for parameter expansion, command substitution, and quote removal.

Next, the shell shall treat this as an arithmetic expression and substitute the value of the expression. The arithmetic expression shall be processed according to the rules given in Section 1.7.2.1 (on page 7), with the following exceptions:

- Only signed long integer arithmetic is required.
- Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in the ISO C standard, Section 6.4.4.1 are required to be recognized as constants.
- The `sizeof()` operator and the prefix and postfix "++" and "--" operators are not required.
- Selection, iteration, and jump statements are not supported.

As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell may use a signed integer type with a rank larger than the rank of **signed long**. The shell may use a real-floating type instead of **signed long** as long as it does not affect the results in cases where there is no overflow. If the expression is invalid, the expansion fails and the shell shall write a message to standard error indicating the failure.

Examples

A simple example using arithmetic expansion:

```
# repeat a command 100 times
x=100
while [ $x -gt 0 ]
do
    command
    x=$((x-1))
done
```

1719 2.6.5 Field Splitting

1720 After parameter expansion (Section 2.6.2 (on page 37)), command substitution (Section 2.6.3 (on
1721 page 40)), and arithmetic expansion (Section 2.6.4 (on page 41)), the shell shall scan the results of
1722 expansions and substitutions that did not occur in double-quotes for field splitting and multiple
1723 fields can result.

1724 The shell shall treat each character of the *IFS* as a delimiter and use the delimiters to split the
1725 results of parameter expansion and command substitution into fields.

1726 1. If the value of *IFS* is a <space>, <tab>, and <newline>, or if it is unset, any sequence of
1727 <space>s, <tab>s, or <newline>s at the beginning or end of the input shall be ignored and
1728 any sequence of those characters within the input shall delimit a field. For example, the
1729 input:

1730 `<newline><space><tab>foo<tab><tab>bar<space>`

1731 yields two fields, **foo** and **bar**.

1732 2. If the value of *IFS* is null, no field splitting shall be performed.

1733 3. Otherwise, the following rules shall be applied in sequence. The term “*IFS* white space” is
1734 used to mean any sequence (zero or more instances) of white space characters that are in
1735 the *IFS* value (for example, if *IFS* contains <space>/<comma>/<tab>, any sequence of
1736 <space>s and <tab>s is considered *IFS* white space).

1737 a. *IFS* white space shall be ignored at the beginning and end of the input.

1738 b. Each occurrence in the input of an *IFS* character that is not *IFS* white space, along
1739 with any adjacent *IFS* white space, shall delimit a field, as described previously.

1740 c. Non-zero-length *IFS* white space shall delimit a field.

1741 2.6.6 Pathname Expansion

1742 After field splitting, if *set -f* is not in effect, each field in the resulting command line shall be
1743 expanded using the algorithm described in Section 2.13 (on page 62), qualified by the rules in
1744 Section 2.13.3 (on page 63).

1745 2.6.7 Quote Removal

1746 The quote characters: ‘\’, ‘\’, and ‘\’ (backslash, single-quote, double-quote) that were
1747 present in the original word shall be removed unless they have themselves been quoted.

2.7 Redirection

Redirection is used to open and close files for the current shell execution environment (see Section 2.12 (on page 61)) or for any command. Redirection operators can be used with numbers representing file descriptors (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.165, File Descriptor) as described below.

The overall format used for redirection is:

```
[n] redir-op word
```

The number *n* is an optional decimal number designating the file descriptor number; the application shall ensure it is delimited from any preceding text and immediately precede the redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the redirection expression. For example:

```
echo \2>a
```

writes the character 2 into file **a**. If any part of *redir-op* is quoted, no redirection expression is recognized. For example:

```
echo 2\>a
```

writes the characters 2>a to standard output. The optional number, redirection operator, and *word* shall not appear in the arguments provided to the command to be executed (if any).

Open files are represented by decimal numbers starting with zero. The largest possible value is implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for use by the application. These numbers are called “file descriptors”. The values 0, 1, and 2 have special meaning and conventional uses and are implied by certain redirection operations; they are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs usually take their input from standard input, and write output on standard output. Error messages are usually written on standard error. The redirection operators can be preceded by one or more digits (with no intervening <blank>s allowed) to designate the file descriptor number.

If the redirection operator is "<<" or "<<–", the word that follows the redirection operator shall be subjected to quote removal; it is unspecified whether any of the other expansions occur. For the other redirection operators, the word that follows the redirection operator shall be subjected to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal. Pathname expansion shall not be performed on the word by a non-interactive shell; an interactive shell may perform it, but shall do so only when the expansion would result in one word.

If more than one redirection operator is specified with a command, the order of evaluation is from beginning to end.

A failure to open or create a file shall cause a redirection to fail.

1784 2.7.1 Redirecting Input

1785 Input redirection shall cause the file whose name results from the expansion of *word* to be
 1786 opened for reading on the designated file descriptor, or standard input if the file descriptor is not
 1787 specified.

1788 The general format for redirecting input is:

1789 `[n] <word`

1790 where the optional *n* represents the file descriptor number. If the number is omitted, the
 1791 redirection shall refer to standard input (file descriptor 0).

1792 2.7.2 Redirecting Output

1793 The two general formats for redirecting output are:

1794 `[n] >word`

1795 `[n] >| word`

1796 where the optional *n* represents the file descriptor number. If the number is omitted, the
 1797 redirection shall refer to standard output (file descriptor 1).

1798 Output redirection using the '*>*' format shall fail if the *noclobber* option is set (see the
 1799 description of *set -C*) and the file named by the expansion of *word* exists and is a regular file.
 1800 Otherwise, redirection using the '*>*' or "*>|*" formats shall cause the file whose name results
 1801 from the expansion of *word* to be created and opened for output on the designated file
 1802 descriptor, or standard output if none is specified. If the file does not exist, it shall be created;
 1803 otherwise, it shall be truncated to be an empty file after being opened.

1804 2.7.3 Appending Redirected Output

1805 Appended output redirection shall cause the file whose name results from the expansion of
 1806 *word* to be opened for output on the designated file descriptor. The file is opened as if the *open()*
 1807 function as defined in the System Interfaces volume of IEEE Std 1003.1-2001 was called with the
 1808 *O_APPEND* flag. If the file does not exist, it shall be created.

1809 The general format for appending redirected output is as follows:

1810 `[n] >>word`

1811 where the optional *n* represents the file descriptor number. If the number is omitted, the
 1812 redirection refers to standard output (file descriptor 1).

1813 2.7.4 Here-Document

1814 The redirection operators "*<<*" and "*<<-*" both allow redirection of lines contained in a shell
 1815 input file, known as a "here-document", to the input of a command.

1816 The here-document shall be treated as a single word that begins after the next *<newline>* and
 1817 continues until there is a line containing only the delimiter and a *<newline>*, with no *<blank>*s in
 1818 between. Then the next here-document starts, if there is one. The format is as follows:

1819 `[n] <<word`
 1820 `here-document`
 1821 `delimiter`

1822 where the optional *n* represents the file descriptor number. If the number is omitted, the here-
 1823 document refers to standard input (file descriptor 0).

If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the *word* itself.

If no characters in *word* are quoted, all lines of the here-document shall be expanded for parameter expansion, command substitution, and arithmetic expansion. In this case, the backslash in the input behaves as the backslash inside double-quotes (see Section 2.2.3 (on page 30)). However, the double-quote character (`"`) shall not be treated specially within a here-document, except when the double-quote appears within `"$ ( ) "`, `"` `"`, or `"${ } "`.

If the redirection symbol is `"<<-"`, all leading `<tab>`s shall be stripped from input lines and the line containing the trailing delimiter. If more than one `"<<"` or `"<<-"` operator is specified on a line, the here-document associated with the first operator shall be supplied first by the application and shall be read first by the shell.

Examples

An example of a here-document follows:

```
cat <<eof1; cat <<eof2
Hi,
eof1
Helene.
eof2
```

2.7.5 Duplicating an Input File Descriptor

The redirection operator:

```
[n] <&word
```

shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a file descriptor already open for input, a redirection error shall result; see Section 2.8.1 (on page 46). If *word* evaluates to `'-'`, file descriptor *n*, or standard input if *n* is not specified, shall be closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word* evaluates to something else, the behavior is unspecified.

2.7.6 Duplicating an Output File Descriptor

The redirection operator:

```
[n] >&word
```

shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a file descriptor already open for output, a redirection error shall result; see Section 2.8.1 (on page 46). If *word* evaluates to `'-'`, file descriptor *n*, or standard output if *n* is not specified, is closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word* evaluates to something else, the behavior is unspecified.

2.7.7 Open File Descriptors for Reading and Writing

The redirection operator:

`[n] <>word`

shall cause the file whose name is the expansion of *word* to be opened for both reading and writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does not exist, it shall be created.

2.8 Exit Status and Errors

2.8.1 Consequences of Shell Errors

For a non-interactive shell, an error condition encountered by a special built-in (see Section 2.14 (on page 64)) or other type of utility shall cause the shell to write a diagnostic message to standard error and exit as shown in the following table:

Error	Special Built-In	Other Utilities
Shell language syntax error	Shall exit	Shall exit
Utility syntax error (option or operand error)	Shall exit	Shall not exit
Redirection error	Shall exit	Shall not exit
Variable assignment error	Shall exit	Shall not exit
Expansion error	Shall exit	Shall exit
Command not found	N / A	May exit
Dot script not found	Shall exit	N / A

An expansion error is one that occurs when the shell expansions defined in Section 2.6 (on page 36) are carried out (for example, "`${x!y}`", because '`!`' is not a valid operator); an implementation may treat these as syntax errors if it is able to detect them during tokenization, rather than during expansion.

If any of the errors shown as “shall exit” or “(may) exit” occur in a subshell, the subshell shall (respectively may) exit with a non-zero status, but the script containing the subshell shall not exit because of the error.

In all of the cases shown in the table, an interactive shell shall write a diagnostic message to standard error without exiting.

2.8.2 Exit Status for Commands

Each command has an exit status that can influence the behavior of other shell commands. The exit status of commands that are not utilities is documented in this section. The exit status of the standard utilities is documented in their respective sections.

If a command is not found, the exit status shall be 127. If the command name is found, but it is not an executable utility, the exit status shall be 126. Applications that invoke utilities without using the shell should use these exit status values to report similar errors.

If a command fails during word expansion or redirection, its exit status shall be greater than zero.

Internally, for purposes of deciding whether a command exits with a non-zero exit status, the shell shall recognize the entire status value retrieved for the command by the equivalent of the `wait()` function `WEXITSTATUS` macro (as defined in the System Interfaces volume of IEEE Std 1003.1-2001). When reporting the exit status with the special parameter '`?`', the shell

shall report the full eight bits of exit status available. The exit status of a command that terminated because it received a signal shall be reported as greater than 128.

2.9 Shell Commands

This section describes the basic structure of shell commands. The following command descriptions each describe a format of the command that is only used to aid the reader in recognizing the command type, and does not formally represent the syntax. Each description discusses the semantics of the command; for a formal definition of the command language, consult Section 2.10 (on page 55).

A *command* is one of the following:

- Simple command (see Section 2.9.1)
- Pipeline (see Section 2.9.2 (on page 49))
- List compound-list (see Section 2.9.3 (on page 50))
- Compound command (see Section 2.9.4 (on page 52))
- Function definition (see Section 2.9.5 (on page 54))

Unless otherwise stated, the exit status of a command shall be that of the last simple command executed by the command. There shall be no limit on the size of any shell command other than that imposed by the underlying system (memory constraints, {ARG_MAX}, and so on).

2.9.1 Simple Commands

A “simple command” is a sequence of optional variable assignments and redirections, in any sequence, optionally followed by words and redirections, terminated by a control operator.

When a given simple command is required to be executed (that is, when any conditional construct such as an AND-OR list or a **case** statement has not bypassed the simple command), the following expansions, assignments, and redirections shall all be performed from the beginning of the command text to the end:

1. The words that are recognized as variable assignments or redirections according to Section 2.10.2 (on page 56) are saved for processing in steps 3 and 4.
2. The words that are not variable assignments or redirections shall be expanded. If any fields remain following their expansion, the first field shall be considered the command name and remaining fields are the arguments for the command.
3. Redirections shall be performed as described in Section 2.7 (on page 43).
4. Each variable assignment shall be expanded for tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal prior to assigning the value.

In the preceding list, the order of steps 3 and 4 may be reversed for the processing of special built-in utilities; see Section 2.14 (on page 64).

If no command name results, variable assignments shall affect the current execution environment. Otherwise, the variable assignments shall be exported for the execution environment of the command and shall not affect the current execution environment (except for special built-ins). If any of the variable assignments attempt to assign a value to a read-only variable, a variable assignment error shall occur. See Section 2.8.1 (on page 46) for the consequences of these errors.

If there is no command name, any redirections shall be performed in a subshell environment; it is unspecified whether this subshell environment is the same one as that used for a command substitution within the command. (To affect the current execution environment, see the *exec* special built-in.) If any of the redirections performed in the current shell execution environment fail, the command shall immediately fail with an exit status greater than zero, and the shell shall write an error message indicating the failure. See Section 2.8.1 (on page 46) for the consequences of these failures on interactive and non-interactive shells.

If there is a command name, execution shall continue as described in Section 2.9.1.1. If there is no command name, but the command contained a command substitution, the command shall complete with the exit status of the last command substitution performed. Otherwise, the command shall complete with a zero exit status.

2.9.1.1 Command Search and Execution

If a simple command results in a command name and an optional list of arguments, the following actions shall be performed:

1. If the command name does not contain any slashes, the first successful step in the following sequence shall occur:
 - a. If the command name matches the name of a special built-in utility, that special built-in utility shall be invoked.
 - b. If the command name matches the name of a function known to this shell, the function shall be invoked as described in Section 2.9.5 (on page 54). If the implementation has provided a standard utility in the form of a function, it shall not be recognized at this point. It shall be invoked in conjunction with the path search in step 1d.
 - c. If the command name matches the name of a utility listed in the following table, that utility shall be invoked.

<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>true</i>	
<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>umask</i>	
<i>command</i>	<i>getopts</i>	<i>pwd</i>	<i>unalias</i>	

- d. Otherwise, the command shall be searched for using the *PATH* environment variable as described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables:

- i. If the search is successful:

- a. If the system has implemented the utility as a regular built-in or as a shell function, it shall be invoked at this point in the path search.
- b. Otherwise, the shell executes the utility in a separate utility environment (see Section 2.12 (on page 61)) with actions equivalent to calling the *execve()* function as defined in the System Interfaces volume of IEEE Std 1003.1-2001 with the *path* argument set to the pathname resulting from the search, *arg0* set to the command name, and the remaining arguments set to the operands, if any.

If the *execve()* function fails due to an error equivalent to the [ENOEXEC] error defined in the System Interfaces volume of IEEE Std 1003.1-2001, the shell shall execute a command equivalent to having a shell invoked with the command name as its first operand, with any remaining arguments

1990 passed to the new shell. If the executable file is not a text file, the shell
 1991 may bypass this command execution. In this case, it shall write an error
 1992 message, and shall return an exit status of 126.

1993 Once a utility has been searched for and found (either as a result of this specific
 1994 search or as part of an unspecified shell start-up activity), an implementation
 1995 may remember its location and need not search for the utility again unless the
 1996 *PATH* variable has been the subject of an assignment. If the remembered
 1997 location fails for a subsequent invocation, the shell shall repeat the search to
 1998 find the new location for the utility, if any.

1999 ii. If the search is unsuccessful, the command shall fail with an exit status of 127
 2000 and the shell shall write an error message.

2001 2. If the command name contains at least one slash, the shell shall execute the utility in a
 2002 separate utility environment with actions equivalent to calling the *execve()* function
 2003 defined in the System Interfaces volume of IEEE Std 1003.1-2001 with the *path* and *arg0*
 2004 arguments set to the command name, and the remaining arguments set to the operands, if
 2005 any.

2006 If the *execve()* function fails due to an error equivalent to the [ENOEXEC] error, the shell
 2007 shall execute a command equivalent to having a shell invoked with the command name as
 2008 its first operand, with any remaining arguments passed to the new shell. If the executable
 2009 file is not a text file, the shell may bypass this command execution. In this case, it shall
 2010 write an error message and shall return an exit status of 126.

2011 2.9.2 Pipelines

2012 A *pipeline* is a sequence of one or more commands separated by the control operator ' | '. The
 2013 standard output of all but the last command shall be connected to the standard input of the next
 2014 command.

2015 The format for a pipeline is:

2016 `[!] command1 [ | command2 ...]`

2017 The standard output of *command1* shall be connected to the standard input of *command2*. The
 2018 standard input, standard output, or both of a command shall be considered to be assigned by the
 2019 pipeline before any redirection specified by redirection operators that are part of the command
 2020 (see Section 2.7 (on page 43)).

2021 If the pipeline is not in the background (see Section 2.9.3.1 (on page 50)), the shell shall wait for
 2022 the last command specified in the pipeline to complete, and may also wait for all commands to
 2023 complete.

2024 Exit Status

2025 If the reserved word ! does not precede the pipeline, the exit status shall be the exit status of the
 2026 last command specified in the pipeline. Otherwise, the exit status shall be the logical NOT of the
 2027 exit status of the last command. That is, if the last command returns zero, the exit status shall be
 2028 1; if the last command returns greater than zero, the exit status shall be zero.

2029 **2.9.3 Lists**

2030 An *AND-OR list* is a sequence of one or more pipelines separated by the operators "&&" and
 2031 "|".

2032 A *list* is a sequence of one or more AND-OR lists separated by the operators ';' and '&' and
 2033 optionally terminated by ';', '&', or <newline>.

2034 The operators "&&" and "||" shall have equal precedence and shall be evaluated with left
 2035 associativity. For example, both of the following commands write solely **bar** to standard output:

```
2036     false && echo foo || echo bar
2037     true  || echo foo && echo bar
```

2038 A ';' or <newline> terminator shall cause the preceding AND-OR list to be executed
 2039 sequentially; an '&' shall cause asynchronous execution of the preceding AND-OR list.

2040 The term “compound-list” is derived from the grammar in Section 2.10 (on page 55); it is
 2041 equivalent to a sequence of *lists*, separated by <newline>s, that can be preceded or followed by
 2042 an arbitrary number of <newline>s.

2043 **Examples**

2044 The following is an example that illustrates <newline>s in compound-lists:

```
2045     while
2046     # a couple of <newline>s
2047     # a list
2048     date && who || ls; cat file
2049     # a couple of <newline>s
2050     # another list
2051     wc file > output & true
2052
2053     do
2054     # 2 lists
2055     ls
2056     cat file
2057     done
```

2057 **2.9.3.1 Asynchronous Lists**

2058 If a command is terminated by the control operator ampersand ('&'), the shell shall execute the
 2059 command asynchronously in a subshell. This means that the shell shall not wait for the
 2060 command to finish before executing the next command.

2061 The format for running a command in the background is:

```
2062     command1 & [command2 & ... ]
```

2063 The standard input for an asynchronous list, before any explicit redirections are performed, shall
 2064 be considered to be assigned to a file that has the same properties as **/dev/null**. If it is an
 2065 interactive shell, this need not happen. In all cases, explicit redirection of standard input shall
 2066 override this activity.

2067 When an element of an asynchronous list (the portion of the list ended by an ampersand, such as
 2068 *command1*, above) is started by the shell, the process ID of the last command in the asynchronous
 2069 list element shall become known in the current shell execution environment; see Section 2.12 (on
 2070 page 61). This process ID shall remain known until:

- 2071 1. The command terminates and the application waits for the process ID.
- 2072 2. Another asynchronous list invoked before "\$!" (corresponding to the previous
- 2073 asynchronous list) is expanded in the current execution environment.

2074 The implementation need not retain more than the {CHILD_MAX} most recent entries in its list

2075 of known process IDs in the current shell execution environment.

2076 **Exit Status**

2077 The exit status of an asynchronous list shall be zero.

2078 2.9.3.2 *Sequential Lists*

2079 Commands that are separated by a semicolon (' ; ') shall be executed sequentially.

2080 The format for executing commands sequentially shall be:

2081 `command1 [ ; command2 ] . . .`

2082 Each command shall be expanded and executed in the order specified.

2083 **Exit Status**

2084 The exit status of a sequential list shall be the exit status of the last command in the list.

2085 2.9.3.3 *AND Lists*

2086 The control operator "&&" denotes an AND list. The format shall be:

2087 `command1 [ && command2 ] . . .`

2088 First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on,

2089 until a command has a non-zero exit status or there are no more commands left to execute. The

2090 commands are expanded only if they are executed.

2091 **Exit Status**

2092 The exit status of an AND list shall be the exit status of the last command that is executed in the

2093 list.

2094 2.9.3.4 *OR Lists*

2095 The control operator " || " denotes an OR List. The format shall be:

2096 `command1 [ || command2 ] . . .`

2097 First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and

2098 so on, until a command has a zero exit status or there are no more commands left to execute.

2099 **Exit Status**

2100 The exit status of an OR list shall be the exit status of the last command that is executed in the

2101 list.

2102 2.9.4 Compound Commands

2103 The shell has several programming constructs that are “compound commands”, which provide
 2104 control flow for commands. Each of these compound commands has a reserved word or control
 2105 operator at the beginning, and a corresponding terminator reserved word or operator at the end.
 2106 In addition, each can be followed by redirections on the same line as the terminator. Each
 2107 redirection shall apply to all the commands within the compound command that do not
 2108 explicitly override that redirection.

2109 2.9.4.1 Grouping Commands

2110 The format for grouping commands is as follows:

2111 (*compound-list*) Execute *compound-list* in a subshell environment; see Section 2.12 (on page
 2112 61). Variable assignments and built-in commands that affect the
 2113 environment shall not remain in effect after the list finishes.

2114 { *compound-list*; } Execute *compound-list* in the current process environment. The semicolon
 2115 shown here is an example of a control operator delimiting the } reserved
 2116 word. Other delimiters are possible, as shown in Section 2.10 (on page
 2117 55); a <newline> is frequently used.

2118 Exit Status

2119 The exit status of a grouping command shall be the exit status of *compound-list*.

2120 2.9.4.2 The for Loop

2121 The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for**
 2122 loop requires that the reserved words **do** and **done** be used to delimit the sequence of
 2123 commands.

2124 The format for the **for** loop is as follows:

```
2125     for name [ in [word ... ] ]
2126     do
2127         compound-list
2128     done
```

2129 First, the list of words following **in** shall be expanded to generate a list of items. Then, the
 2130 variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no
 2131 items result from the expansion, the *compound-list* shall not be executed. Omitting:

```
2132     in word...
```

2133 shall be equivalent to:

```
2134     in "$@"
```

2135 Exit Status

2136 The exit status of a **for** command shall be the exit status of the last command that executes. If
 2137 there are no items, the exit status shall be zero.

2138 2.9.4.3 *Case Conditional Construct*

2139 The conditional construct **case** shall execute the *compound-list* corresponding to the first one of
 2140 several *patterns* (see Section 2.13 (on page 62)) that is matched by the string resulting from the
 2141 tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote
 2142 removal of the given word. The reserved word **in** shall denote the beginning of the patterns to be
 2143 matched. Multiple patterns with the same *compound-list* shall be delimited by the ' | ' symbol.
 2144 The control operator ') ' terminates a list of patterns corresponding to a given action. The
 2145 *compound-list* for each list of patterns, with the possible exception of the last, shall be terminated
 2146 with " ; ; ". The **case** construct terminates with the reserved word **esac** (**case** reversed).

2147 The format for the **case** construct is as follows:

```
2148     case word in
2149         [(]pattern1) compound-list;;
2150         [(]pattern[ | pattern] ... ) compound-list;;] ...
2151         [(]pattern[ | pattern] ... ) compound-list]
2152     esac
```

2153 The " ; ; " is optional for the last *compound-list*.

2154 In order from the beginning to the end of the **case** statement, each *pattern* that labels a
 2155 *compound-list* shall be subjected to tilde expansion, parameter expansion, command substitution,
 2156 and arithmetic expansion, and the result of these expansions shall be compared against the
 2157 expansion of *word*, according to the rules described in Section 2.13 (on page 62) (which also
 2158 describes the effect of quoting parts of the pattern). After the first match, no more patterns shall
 2159 be expanded, and the *compound-list* shall be executed. The order of expansion and comparison of
 2160 multiple *patterns* that label a *compound-list* statement is unspecified.

2161 **Exit Status**

2162 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be
 2163 the exit status of the last command executed in the *compound-list*.

2164 2.9.4.4 *The if Conditional Construct*

2165 The **if** command shall execute a *compound-list* and use its exit status to determine whether to
 2166 execute another *compound-list*.

2167 The format for the **if** construct is as follows:

```
2168     if compound-list
2169     then
2170         compound-list
2171     [elif compound-list
2172     then
2173         compound-list] ...
2174     [else
2175         compound-list]
2176     fi
```

2177 The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be
 2178 executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed,
 2179 in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command
 2180 shall complete. Otherwise, the **else** *compound-list* shall be executed.

2181 **Exit Status**

2182 The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that
 2183 was executed, or zero, if none was executed.

2184 2.9.4.5 *The while Loop*

2185 The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has
 2186 a zero exit status.

2187 The format of the **while** loop is as follows:

```
2188            while compound-list-1
2189            do
2190                compound-list-2
2191            done
```

2192 The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command
 2193 shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

2194 **Exit Status**

2195 The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or
 2196 zero if none was executed.

2197 2.9.4.6 *The until Loop*

2198 The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has
 2199 a non-zero exit status.

2200 The format of the **until** loop is as follows:

```
2201            until compound-list-1
2202            do
2203                compound-list-2
2204            done
```

2205 The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command
 2206 completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

2207 **Exit Status**

2208 The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or
 2209 zero if none was executed.

2210 2.9.5 **Function Definition Command**

2211 A function is a user-defined name that is used as a simple command to call a compound
 2212 command with new positional parameters. A function is defined with a “function definition
 2213 command”.

2214 The format of a function definition command is as follows:

```
2215            fname() compound-command[io-redirect ...]
```

2216 The function is named *fname*; the application shall ensure that it is a name (see the Base
 2217 Definitions volume of IEEE Std 1003.1-2001, Section 3.230, Name). An implementation may
 2218 allow other characters in a function name as an extension. The implementation shall maintain
 2219 separate name spaces for functions and variables.

2220 The argument *compound-command* represents a compound command, as described in Section
2221 2.9.4 (on page 52).

2222 When the function is declared, none of the expansions in Section 2.6 (on page 36) shall be
2223 performed on the text in *compound-command* or *io-redirect*; all expansions shall be performed as
2224 normal each time the function is called. Similarly, the optional *io-redirect* redirections and any
2225 variable assignments within *compound-command* shall be performed during the execution of the
2226 function itself, not the function definition. See Section 2.8.1 (on page 46) for the consequences of
2227 failures of these operations on interactive and non-interactive shells.

2228 When a function is executed, it shall have the syntax-error and variable-assignment properties
2229 described for special built-in utilities in the enumerated list at the beginning of Section 2.14 (on
2230 page 64).

2231 The *compound-command* shall be executed whenever the function name is specified as the name
2232 of a simple command (see Section 2.9.1.1 (on page 48)). The operands to the command
2233 temporarily shall become the positional parameters during the execution of the *compound-*
2234 *command*; the special parameter '#' also shall be changed to reflect the number of operands. The
2235 special parameter 0 shall be unchanged. When the function completes, the values of the
2236 positional parameters and the special parameter '#' shall be restored to the values they had
2237 before the function was executed. If the special built-in *return* is executed in the *compound-*
2238 *command*, the function completes and execution shall resume with the next command after the
2239 function call.

2240 **Exit Status**

2241 The exit status of a function definition shall be zero if the function was declared successfully;
2242 otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit
2243 status of the last command executed by the function.

2244 **2.10 Shell Grammar**

2245 The following grammar defines the Shell Command Language. This formal syntax shall take
2246 precedence over the preceding text syntax description.

2247 **2.10.1 Shell Grammar Lexical Conventions**

2248 The input language to the shell must be first recognized at the character level. The resulting
2249 tokens shall be classified by their immediate context according to the following rules (applied in
2250 order). These rules shall be used to determine what a “token” is that is subject to parsing at the
2251 token level. The rules for token recognition in Section 2.3 (on page 31) shall apply.

- 2252 1. A <newline> shall be returned as the token identifier **NEWLINE**.
- 2253 2. If the token is an operator, the token identifier for that operator shall result.
- 2254 3. If the string consists solely of digits and the delimiter character is one of '<' or '>', the
2255 token identifier **IO_NUMBER** shall be returned.
- 2256 4. Otherwise, the token identifier **TOKEN** results.

2257 Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields
2258 **WORD**, a **NAME**, an **ASSIGNMENT**, or one of the reserved words below, dependent upon the
2259 context. Some of the productions in the grammar below are annotated with a rule number from
2260 the following list. When a **TOKEN** is seen where one of those annotated productions could be
2261 used to reduce the symbol, the applicable rule shall be applied to convert the token identifier

type of the **TOKEN** to a token identifier acceptable at that point in the grammar. The reduction shall then proceed based upon the token identifier type yielded by the rule applied. When more than one rule applies, the highest numbered rule shall apply (which in turn may refer to another rule). (Note that except in rule 7, the presence of an '=' in the token has no effect.)

The **WORD** tokens shall have the word expansion rules applied to them immediately before the associated command is executed, not at the time the command is parsed.

2.10.2 Shell Grammar Rules

1. [Command Name]

When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any state where only a reserved word could be the next correct token, proceed as above.

Note: Because at this point quote marks are retained in the token, quoted strings cannot be recognized as reserved words. This rule also implies that reserved words are not recognized except in certain positions in the input, such as after a <newline> or semicolon; the grammar presumes that if the reserved word is intended, it is properly delimited by the user, and does not attempt to reflect that requirement directly. Also note that line joining is done before tokenization, as described in Section 2.2.1 (on page 30), so escaped <newline>s are already removed at this point.

Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or applies globally.

2. [Redirection to or from filename]

The expansions specified in Section 2.7 (on page 43) shall occur. As specified there, exactly one field can result (or the result is unspecified), and there are additional requirements on pathname expansion.

3. [Redirection from here-document]

Quote removal shall be applied to the word to determine the delimiter that is used to find the end of the here-document that begins after the next <newline>.

4. [Case statement termination]

When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall result. Otherwise, the token **WORD** shall be returned.

5. [NAME in for]

When the **TOKEN** meets the requirements for a name (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.230, Name), the token identifier **NAME** shall result. Otherwise, the token **WORD** shall be returned.

6. [Third word of for and case]

a. [case only]

When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall result. Otherwise, the token **WORD** shall be returned.

b. [for only]

When the **TOKEN** is exactly the reserved word **in** or **do**, the token identifier for **in** or **do** shall result, respectively. Otherwise, the token **WORD** shall be returned.

2303 (For a. and b.: As indicated in the grammar, a *linebreak* precedes the tokens **in** and **do**. If
 2304 <newline>s are present at the indicated location, it is the token after them that is treated in
 2305 this fashion.)

2306 7. [Assignment preceding command name]

2307 a. [When the first word]

2308 If the **TOKEN** does not contain the character '=', rule 1 is applied. Otherwise, 7b
 2309 shall be applied.

2310 b. [Not the first word]

2311 If the **TOKEN** contains the equal sign character:

2312 — If it begins with '=', the token **WORD** shall be returned.

2313 — If all the characters preceding '=' form a valid name (see the Base Definitions
 2314 volume of IEEE Std 1003.1-2001, Section 3.230, Name), the token
 2315 **ASSIGNMENT_WORD** shall be returned. (Quoted characters cannot participate
 2316 in forming a valid name.)

2317 — Otherwise, it is unspecified whether it is **ASSIGNMENT_WORD** or **WORD** that
 2318 is returned.

2319 Assignment to the **NAME** shall occur as specified in Section 2.9.1 (on page 47).

2320 8. [**NAME** in function]

2321 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word
 2322 shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token
 2323 identifier **NAME** shall result. Otherwise, rule 7 applies.

2324 9. [Body of function]

2325 Word expansion and assignment shall never occur, even when required by the rules above,
 2326 when this rule is being parsed. Each **TOKEN** that might either be expanded or have
 2327 assignment applied to it shall instead be returned as a single **WORD** consisting only of
 2328 characters that are exactly the token described in Section 2.3 (on page 31).

2329 /* -----
 2330 The grammar symbols
 2331 ----- */

2332 %token WORD
 2333 %token ASSIGNMENT_WORD
 2334 %token NAME
 2335 %token NEWLINE
 2336 %token IO_NUMBER

2337 /* The following are the operators mentioned above. */

2338 %token AND_IF OR_IF DSEMI
 2339 /* '&&' '|'| ';' */

2340 %token DLESS DGREAT LESSAND GREATAND LESSGREAT DLESSDASH
 2341 /* '<<' '>>' '<&' '>&' '<>' '<<-' */

2342 %token CLOBBER
 2343 /* '>|' */

```

2344      /* The following are the reserved words. */
2345      %token  If      Then      Else      Elif      Fi      Do      Done
2346      /*      'if'    'then'    'else'    'elif'    'fi'    'do'    'done'    */

2347      %token  Case     Esac      While      Until      For
2348      /*      'case'   'esac'    'while'    'until'    'for'    */

2349      /* These are reserved words, not operator tokens, and are
2350         recognized when reserved words are recognized. */

2351      %token  Lbrace    Rbrace      Bang
2352      /*      '{'      '}'        '!'        */

2353      %token  In
2354      /*      'in'     */

2355      /* -----
2356         The Grammar
2357      ----- */

2358      %start  complete_command
2359      %%
2360      complete_command : list separator
2361                       | list
2362                       ;
2363      list             : list separator_op and_or
2364                       |                               and_or
2365                       ;
2366      and_or           :                               pipeline
2367                       | and_or AND_IF linebreak pipeline
2368                       | and_or OR_IF  linebreak pipeline
2369                       ;
2370      pipeline         :      pipe_sequence
2371                       | Bang pipe_sequence
2372                       ;
2373      pipe_sequence    :                               command
2374                       | pipe_sequence '|' linebreak command
2375                       ;
2376      command          : simple_command
2377                       | compound_command
2378                       | compound_command redirect_list
2379                       | function_definition
2380                       ;
2381      compound_command : brace_group
2382                       | subshell
2383                       | for_clause
2384                       | case_clause
2385                       | if_clause
2386                       | while_clause
2387                       | until_clause
2388                       ;
2389      subshell         : '(' compound_list ')'
2390                       ;
2391      compound_list     :                               term
2392                       | newline_list term

```

```

2393             |           term separator
2394             | newline_list term separator
2395             ;
2396 term          : term separator and_or
2397             |           and_or
2398             ;
2399 for_clause    : For name linebreak                               do_group
2400             | For name linebreak in                               sequential_sep do_group
2401             | For name linebreak in wordlist sequential_sep do_group
2402             ;
2403 name          : NAME                                           /* Apply rule 5 */
2404             ;
2405 in            : In                                           /* Apply rule 6 */
2406             ;
2407 wordlist      : wordlist WORD
2408             |           WORD
2409             ;
2410 case_clause   : Case WORD linebreak in linebreak case_list      Esac
2411             | Case WORD linebreak in linebreak case_list_ns Esac
2412             | Case WORD linebreak in linebreak                  Esac
2413             ;
2414 case_list_ns  : case_list case_item_ns
2415             |           case_item_ns
2416             ;
2417 case_list     : case_list case_item
2418             |           case_item
2419             ;
2420 case_item_ns  :           pattern ')' linebreak
2421             |           pattern ')' compound_list linebreak
2422             | '(' pattern ')' linebreak
2423             | '(' pattern ')' compound_list linebreak
2424             ;
2425 case_item     :           pattern ')' linebreak DSEMI linebreak
2426             |           pattern ')' compound_list DSEMI linebreak
2427             | '(' pattern ')' linebreak DSEMI linebreak
2428             | '(' pattern ')' compound_list DSEMI linebreak
2429             ;
2430 pattern       :           WORD                               /* Apply rule 4 */
2431             | pattern '|' WORD                               /* Do not apply rule 4 */
2432             ;
2433 if_clause     : If compound_list Then compound_list else_part Fi
2434             | If compound_list Then compound_list              Fi
2435             ;
2436 else_part     : Elif compound_list Then else_part
2437             | Else compound_list
2438             ;
2439 while_clause  : While compound_list do_group
2440             ;
2441 until_clause  : Until compound_list do_group
2442             ;
2443 function_definition : fname '(' ')' linebreak function_body
2444             ;

```

```

2445     function_body      : compound_command          /* Apply rule 9 */
2446                        | compound_command redirect_list /* Apply rule 9 */
2447                        ;
2448     fname               : NAME                      /* Apply rule 8 */
2449                        ;
2450     brace_group         : Lbrace compound_list Rbrace
2451                        ;
2452     do_group            : Do compound_list Done      /* Apply rule 6 */
2453                        ;
2454     simple_command      : cmd_prefix cmd_word cmd_suffix
2455                        | cmd_prefix cmd_word
2456                        | cmd_prefix
2457                        | cmd_name cmd_suffix
2458                        | cmd_name
2459                        ;
2460     cmd_name            : WORD                      /* Apply rule 7a */
2461                        ;
2462     cmd_word            : WORD                      /* Apply rule 7b */
2463                        ;
2464     cmd_prefix          :
2465                        | io_redirect
2466                        | cmd_prefix io_redirect
2467                        | ASSIGNMENT_WORD
2468                        | cmd_prefix ASSIGNMENT_WORD
2469                        ;
2470     cmd_suffix          :
2471                        | io_redirect
2472                        | cmd_suffix io_redirect
2473                        | WORD
2474                        | cmd_suffix WORD
2475                        ;
2476     redirect_list      :
2477                        | io_redirect
2478                        | redirect_list io_redirect
2479                        ;
2480     io_redirect         :
2481                        | io_file
2482                        | IO_NUMBER io_file
2483                        | io_here
2484                        | IO_NUMBER io_here
2485                        ;
2486     io_file            :
2487                        | '<' filename
2488                        | LESSAND filename
2489                        | '>' filename
2490                        | GREATAND filename
2491                        | DGREAT filename
2492                        | LESSGREAT filename
2493                        | CLOBBER filename
2494                        ;
2495     filename           : WORD                      /* Apply rule 2 */
2496                        ;
2497     io_here            : DLESS here_end
2498                        | DLESSDASH here_end
2499                        ;
2500     here_end           : WORD                      /* Apply rule 3 */
2501                        ;

```



```

2497     newline_list      :                NEWLINE
2498                       | newline_list NEWLINE
2499                       ;
2500     linebreak          : newline_list
2501                       | /* empty */
2502                       ;
2503     separator_op       : '&'
2504                       | ';'
2505                       ;
2506     separator          : separator_op linebreak
2507                       | newline_list
2508                       ;
2509     sequential_sep     : ';' linebreak
2510                       | newline_list
2511                       ;

```

2512 2.11 Signals and Error Handling

2513 When a command is in an asynchronous list, the shell shall prevent SIGQUIT and SIGINT
 2514 signals from the keyboard from interrupting the command. Otherwise, signals shall have the
 2515 values inherited by the shell from its parent (see also the *trap* special built-in).

2516 When a signal for which a trap has been set is received while the shell is waiting for the
 2517 completion of a utility executing a foreground command, the trap associated with that signal
 2518 shall not be executed until after the foreground command has completed. When the shell is
 2519 waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a
 2520 signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit
 2521 status >128, immediately after which the trap associated with that signal shall be taken.

2522 If multiple signals are pending for the shell for which there are associated trap actions, the order
 2523 of execution of trap actions is unspecified.

2524 2.12 Shell Execution Environment

2525 A shell execution environment consists of the following:

- 2526 • Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- 2527 • Working directory as set by *cd*
- 2528 • File creation mask set by *umask*
- 2529 • Current traps set by *trap*
- 2530 • Shell parameters that are set by variable assignment (see the *set* special built-in) or from the
 2531 System Interfaces volume of IEEE Std 1003.1-2001 environment inherited by the shell when it
 2532 begins (see the *export* special built-in)
- 2533 • Shell functions; see Section 2.9.5 (on page 54)
- 2534 • Options turned on at invocation or by *set*
- 2535 • Process IDs of the last commands in asynchronous lists known to this shell environment; see
 2536 Section 2.9.3.1 (on page 50)

- Shell aliases; see Section 2.3.1 (on page 32)

Utilities other than the special built-ins (see Section 2.14 (on page 64)) shall be invoked in a separate environment that consists of the following. The initial value of these objects shall be the same as that for the parent shell, except as noted below.

- Open files inherited on invocation of the shell, open files controlled by the *exec* special built-in plus any modifications, and additions specified by any redirections to the utility
- Current working directory
- File creation mask
- If the utility is a shell script, traps caught by the shell shall be set to the default values and traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell script, the trap actions (default or ignore) shall be mapped into the appropriate signal handling actions for the utility
- Variables with the *export* attribute, along with those explicitly exported for the duration of the command, shall be passed to the utility environment variables

The environment of the shell process shall not be changed by the utility unless explicitly specified by the utility description (for example, *cd* and *umask*).

A subshell environment shall be created as a duplicate of the shell environment, except that signal traps set by that shell environment shall be set to the default values. Changes made to the subshell environment shall not affect the shell environment. Command substitution, commands that are grouped with parentheses, and asynchronous lists shall be executed in a subshell environment. Additionally, each command of a multi-command pipeline is in a subshell environment; as an extension, however, any or all commands in a pipeline may be executed in the current environment. All other commands shall be executed in the current shell environment.

2.13 Pattern Matching Notation

The pattern matching notation described in this section is used to specify patterns for matching strings in the shell. Historically, pattern matching notation is related to, but slightly different from, the regular expression notation described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular Expressions. For this reason, the description of the rules for this pattern matching notation are based on the description of regular expression notation, modified to include backslash escape processing.

2.13.1 Patterns Matching a Single Character

The following patterns matching a single character shall match a single character: ordinary characters, special pattern characters, and pattern bracket expressions. The pattern bracket expression also shall match a single collating element. A backslash character shall escape the following character. The escaping backslash shall be discarded.

An ordinary character is a pattern that shall match itself. It can be any character in the supported character set except for NUL, those special shell characters in Section 2.2 (on page 30) that require quoting, and the following three special pattern characters. Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. If any character (ordinary, shell special, or pattern special) is quoted, that pattern shall match the character itself. The shell special characters always require quoting.

2579 When unquoted and outside a bracket expression, the following three characters shall have
 2580 special meaning in the specification of patterns:

2581 ? A question-mark is a pattern that shall match any character.

2582 * An asterisk is a pattern that shall match multiple characters, as described in Section 2.13.2.

2583 [The open bracket shall introduce a pattern bracket expression.

2584 The description of basic regular expression bracket expressions in the Base Definitions volume
 2585 of IEEE Std 1003.1-2001, Section 9.3.5, RE Bracket Expression shall also apply to the pattern
 2586 bracket expression, except that the exclamation mark character ('!') shall replace the
 2587 circumflex character ('^') in its role in a “non-matching list” in the regular expression notation.
 2588 A bracket expression starting with an unquoted circumflex character produces unspecified
 2589 results.

2590 When pattern matching is used where shell quote removal is not performed (such as in the
 2591 argument to the *find -name* primary when *find* is being called using one of the *exec* functions as
 2592 defined in the System Interfaces volume of IEEE Std 1003.1-2001, or in the *pattern* argument to
 2593 the *fnmatch()* function), special characters can be escaped to remove their special meaning by
 2594 preceding them with a backslash character. This escaping backslash is discarded. The sequence
 2595 "\\\" represents one literal backslash. All of the requirements and effects of quoting on ordinary,
 2596 shell special, and special pattern characters shall apply to escaping in this context.

2597 **2.13.2 Patterns Matching Multiple Characters**

2598 The following rules are used to construct patterns matching multiple characters from patterns
 2599 matching a single character:

- 2600 1. The asterisk ('*') is a pattern that shall match any string, including the null string.
- 2601 2. The concatenation of patterns matching a single character is a valid pattern that shall
 2602 match the concatenation of the single characters or collating elements matched by each of
 2603 the concatenated patterns.
- 2604 3. The concatenation of one or more patterns matching a single character with one or more
 2605 asterisks is a valid pattern. In such patterns, each asterisk shall match a string of zero or
 2606 more characters, matching the greatest possible number of characters that still allows the
 2607 remainder of the pattern to match the string.

2608 **2.13.3 Patterns Used for Filename Expansion**

2609 The rules described so far in Section 2.13.1 (on page 62) and Section 2.13.2 are qualified by the
 2610 following rules that apply when pattern matching notation is used for filename expansion:

- 2611 1. The slash character in a pathname shall be explicitly matched by using one or more slashes
 2612 in the pattern; it shall neither be matched by the asterisk or question-mark special
 2613 characters nor by a bracket expression. Slashes in the pattern shall be identified before
 2614 bracket expressions; thus, a slash cannot be included in a pattern bracket expression used
 2615 for filename expansion. If a slash character is found following an unescaped open square
 2616 bracket character before a corresponding closing square bracket is found, the open bracket
 2617 shall be treated as an ordinary character. For example, the pattern "a[b/c]d" does not
 2618 match such pathnames as **abd** or **a/d**. It only matches a pathname of literally **a[b/c]d**.
- 2619 2. If a filename begins with a period ('.'), the period shall be explicitly matched by using a
 2620 period as the first character of the pattern or immediately following a slash character. The
 2621 leading period shall not be matched by:

- The asterisk or question-mark special characters
- A bracket expression containing a non-matching list, such as "[!a]", a range expression, such as "[%–0]", or a character class expression, such as "[[:punct:]]"

It is unspecified whether an explicit period in a bracket expression matching list, such as "[.abc]", can match a leading period in a filename.

3. Specified patterns shall be matched against existing filenames and pathnames, as appropriate. Each component that contains a pattern character shall require read permission in the directory containing that component. Any component, except the last, that does not contain a pattern character shall require search permission. For example, given the pattern:

```
/foo/bar/x*/bam
```

search permission is needed for directories `/` and `foo`, search and read permissions are needed for directory `bar`, and search permission is needed for each `x*` directory. If the pattern matches any existing filenames or pathnames, the pattern shall be replaced with those filenames and pathnames, sorted according to the collating sequence in effect in the current locale. If the pattern contains an invalid bracket expression or does not match any existing filenames or pathnames, the pattern string shall be left unchanged.

2.14 Special Built-In Utilities

The following “special built-in” utilities shall be supported in the shell command language. The output of each command, if any, shall be written to standard output, subject to the normal redirection and piping possible with all commands.

The term “built-in” implies that the shell can execute the utility directly and does not need to search for it. An implementation may choose to make any utility a built-in; however, the special built-in utilities described here differ from regular built-in utilities in two respects:

1. A syntax error in a special built-in utility may cause a shell executing that utility to abort, while a syntax error in a regular built-in utility shall not cause a shell executing that utility to abort. (See Section 2.8.1 (on page 46) for the consequences of errors on interactive and non-interactive shells.) If a special built-in utility encountering a syntax error does not abort the shell, its exit value shall be non-zero.
2. Variable assignments specified with special built-in utilities remain in effect after the built-in completes; this shall not be the case with a regular built-in or other utility.

The special built-in utilities in this section need not be provided in a manner accessible via the `exec` family of functions defined in the System Interfaces volume of IEEE Std 1003.1-2001.

Some of the special built-ins are described as conforming to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines. For those that are not, the requirement in Section 1.11 (on page 20) that “—” be recognized as a first argument to be discarded does not apply and a conforming application shall not use that argument.

2659 **NAME**

2660 break — exit from for, while, or until loop

2661 **SYNOPSIS**2662 break [*n*]2663 **DESCRIPTION**

2664 The *break* utility shall exit from the smallest enclosing **for**, **while**, or **until** loop, if any; or from the
 2665 *n*th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer greater than or
 2666 equal to 1. The default shall be equivalent to *n*=1. If *n* is greater than the number of enclosing
 2667 loops, the outermost enclosing loop shall be exited. Execution shall continue with the command
 2668 immediately following the loop.

2669 **OPTIONS**

2670 None.

2671 **OPERANDS**

2672 See the DESCRIPTION.

2673 **STDIN**

2674 Not used.

2675 **INPUT FILES**

2676 None.

2677 **ENVIRONMENT VARIABLES**

2678 None.

2679 **ASYNCHRONOUS EVENTS**

2680 Default.

2681 **STDOUT**

2682 Not used.

2683 **STDERR**

2684 The standard error shall be used only for diagnostic messages.

2685 **OUTPUT FILES**

2686 None.

2687 **EXTENDED DESCRIPTION**

2688 None.

2689 **EXIT STATUS**

2690 0 Successful completion.

2691 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.2692 **CONSEQUENCES OF ERRORS**

2693 Default.

2694 APPLICATION USAGE

2695 None.

2696 EXAMPLES

```
2697       for i in * do
2698           if test -d "$i" then break fi done
```

2699 RATIONALE

2700 In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer
2701 to a label associated with the appropriate loop as a preferable alternative to the *n* method.
2702 However, this volume of IEEE Std 1003.1-2001 does reserve the name space of command names
2703 ending with a colon. It is anticipated that a future implementation could take advantage of this
2704 and provide something like:

```
2705       outofloop: for i in a b c d e
2706       do
2707           for j in 0 1 2 3 4 5 6 7 8 9
2708           do
2709               if test -r "${i}${j}"
2710               then break outofloop
2711               fi
2712           done
2713       done
```

2714 and that this might be standardized after implementation experience is achieved.

2715 FUTURE DIRECTIONS

2716 None.

2717 SEE ALSO

2718 Section 2.14 (on page 64)

2719 CHANGE HISTORY**2720 Issue 6**

2721 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
2722 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
2723 behavior is intended.

2724 **NAME**

2725 colon — null utility

2726 **SYNOPSIS**2727 : [*argument* ...]2728 **DESCRIPTION**

2729 This utility shall only expand command *arguments*. It is used when a command is needed, as in
2730 the **then** condition of an **if** command, but nothing is to be done by the command.

2731 **OPTIONS**

2732 None.

2733 **OPERANDS**

2734 See the DESCRIPTION. |

2735 **STDIN**

2736 Not used. |

2737 **INPUT FILES**

2738 None.

2739 **ENVIRONMENT VARIABLES**

2740 None.

2741 **ASYNCHRONOUS EVENTS**

2742 Default. |

2743 **STDOUT**

2744 Not used. |

2745 **STDERR**

2746 The standard error shall be used only for diagnostic messages. |

2747 **OUTPUT FILES**

2748 None.

2749 **EXTENDED DESCRIPTION**

2750 None.

2751 **EXIT STATUS**

2752 Zero.

2753 **CONSEQUENCES OF ERRORS**

2754 Default. |

2755 **APPLICATION USAGE**

2756 None.

2757 **EXAMPLES**

```
2758       : ${X=abc}  
2759       if       false  
2760       then     :  
2761       else     echo $X  
2762       fi  
2763       abc
```

2764 As with any of the special built-ins, the null utility can also have variable assignments and
2765 redirections associated with it, such as:

2766 `x=y : > z`

2767 which sets variable `x` to the value `y` (so that it persists after the null utility completes) and creates
2768 or truncates file `z`.

2769 **RATIONALE**

2770 None.

2771 **FUTURE DIRECTIONS**

2772 None.

2773 **SEE ALSO**

2774 Section 2.14 (on page 64)

2775 **CHANGE HISTORY**

2776 **Issue 6**

2777 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
2778 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
2779 behavior is intended.

2780 **NAME**

2781 continue — continue for, while, or until loop

2782 **SYNOPSIS**2783 continue [*n*]2784 **DESCRIPTION**

2785 The *continue* utility shall return to the top of the smallest enclosing **for**, **while**, or **until** loop, or to
 2786 the top of the *n*th enclosing loop, if *n* is specified. This involves repeating the condition list of a
 2787 **while** or **until** loop or performing the next assignment of a **for** loop, and re-executing the loop if
 2788 appropriate.

2789 The value of *n* is a decimal integer greater than or equal to 1. The default shall be equivalent to
 2790 *n*=1. If *n* is greater than the number of enclosing loops, the outermost enclosing loop shall be
 2791 used.

2792 **OPTIONS**

2793 None.

2794 **OPERANDS**

2795 See the DESCRIPTION.

2796 **STDIN**

2797 Not used.

2798 **INPUT FILES**

2799 None.

2800 **ENVIRONMENT VARIABLES**

2801 None.

2802 **ASYNCHRONOUS EVENTS**

2803 Default.

2804 **STDOUT**

2805 Not used.

2806 **STDERR**

2807 The standard error shall be used only for diagnostic messages.

2808 **OUTPUT FILES**

2809 None.

2810 **EXTENDED DESCRIPTION**

2811 None.

2812 **EXIT STATUS**

2813 0 Successful completion.

2814 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.2815 **CONSEQUENCES OF ERRORS**

2816 Default.

2817 **APPLICATION USAGE**

2818 None.

2819 **EXAMPLES**

```
2820       for i in *
2821       do
2822           if test -d "$i"
2823           then continue
2824           fi
2825           echo "\"$i\" \" is not a directory.
2826       done
```

2827 **RATIONALE**

2828 None.

2829 **FUTURE DIRECTIONS**

2830 None.

2831 **SEE ALSO**

2832 Section 2.14 (on page 64)

2833 **CHANGE HISTORY**2834 **Issue 6**

```
2835       IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
2836       sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
2837       behavior is intended.
```

2838	NAME		
2839		dot — execute commands in the current environment	
2840	SYNOPSIS		
2841		. <i>file</i>	
2842	DESCRIPTION		
2843		The shell shall execute commands from the <i>file</i> in the current environment.	
2844		If <i>file</i> does not contain a slash, the shell shall use the search path specified by <i>PATH</i> to find the	
2845		directory containing <i>file</i> . Unlike normal command search, however, the file searched for by the	
2846		<i>dot</i> utility need not be executable. If no readable file is found, a non-interactive shell shall abort;	
2847		an interactive shell shall write a diagnostic message to standard error, but this condition shall	
2848		not be considered a syntax error.	
2849	OPTIONS		
2850		None.	
2851	OPERANDS		
2852		See the DESCRIPTION.	
2853	STDIN		
2854		Not used.	
2855	INPUT FILES		
2856		See the DESCRIPTION.	
2857	ENVIRONMENT VARIABLES		
2858		See the DESCRIPTION.	
2859	ASYNCHRONOUS EVENTS		
2860		Default.	
2861	STDOUT		
2862		Not used.	
2863	STDERR		
2864		The standard error shall be used only for diagnostic messages.	
2865	OUTPUT FILES		
2866		None.	
2867	EXTENDED DESCRIPTION		
2868		None.	
2869	EXIT STATUS		
2870		Returns the value of the last command executed, or a zero exit status if no command is executed.	
2871	CONSEQUENCES OF ERRORS		
2872		Default.	

2873 **APPLICATION USAGE**

2874 None.

2875 **EXAMPLES**

2876 cat foobar

2877 **foo=hello bar=world**

2878 . foobar

2879 echo \$foo \$bar

2880 **hello world**2881 **RATIONALE**

2882 Some older implementations searched the current directory for the *file*, even if the value of *PATH*
2883 disallowed it. This behavior was omitted from this volume of IEEE Std 1003.1-2001 due to
2884 concerns about introducing the susceptibility to trojan horses that the user might be trying to
2885 avoid by leaving **dot** out of *PATH*.

2886 The KornShell version of *dot* takes optional arguments that are set to the positional parameters.

2887 This is a valid extension that allows a *dot* script to behave identically to a function.

2888 **FUTURE DIRECTIONS**

2889 None.

2890 **SEE ALSO**

2891 Section 2.14 (on page 64)

2892 **CHANGE HISTORY**2893 **Issue 6**

2894 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
2895 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
2896 behavior is intended.

2897 **NAME**

2898 eval — construct command by concatenating arguments

2899 **SYNOPSIS**2900 eval [*argument* ...]2901 **DESCRIPTION**2902 The *eval* utility shall construct a command by concatenating *arguments* together, separating each
2903 with a <space>. The constructed command shall be read and executed by the shell.2904 **OPTIONS**

2905 None.

2906 **OPERANDS**

2907 See the DESCRIPTION. |

2908 **STDIN**

2909 Not used. |

2910 **INPUT FILES**

2911 None.

2912 **ENVIRONMENT VARIABLES**

2913 None.

2914 **ASYNCHRONOUS EVENTS**

2915 Default. |

2916 **STDOUT**

2917 Not used. |

2918 **STDERR**

2919 The standard error shall be used only for diagnostic messages. |

2920 **OUTPUT FILES**

2921 None.

2922 **EXTENDED DESCRIPTION**

2923 None.

2924 **EXIT STATUS**2925 If there are no *arguments*, or only null arguments, *eval* shall return a zero exit status; otherwise, it
2926 shall return the exit status of the command defined by the string of concatenated *arguments*
2927 separated by <space>s.2928 **CONSEQUENCES OF ERRORS**

2929 Default. |

2930 **APPLICATION USAGE**

2931 None.

2932 **EXAMPLES**

2933 foo=10 x=foo

2934 y=' \$ ' \$x

2935 echo \$y

2936 \$foo

2937 eval y=' \$ ' \$x

2938 echo \$y

2939 10

2940 **RATIONALE**

2941 None.

2942 **FUTURE DIRECTIONS**

2943 None.

2944 **SEE ALSO**

2945 Section 2.14 (on page 64)

2946 **CHANGE HISTORY**2947 **Issue 6**

2948 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
2949 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
2950 behavior is intended.

2951 **NAME**

2952 exec — execute commands and open, close, or copy file descriptors

2953 **SYNOPSIS**2954 exec [*command* [*argument* ...]]2955 **DESCRIPTION**2956 The *exec* utility shall open, close, and/or copy file descriptors as specified by any redirections as
2957 part of the command.2958 If *exec* is specified without *command* or *arguments*, and any file descriptors with numbers greater
2959 than 2 are opened with associated redirection statements, it is unspecified whether those file
2960 descriptors remain open when the shell invokes another utility. Scripts concerned that child
2961 shells could misuse open file descriptors can always close them explicitly, as shown in one of the
2962 following examples.2963 If *exec* is specified with *command*, it shall replace the shell with *command* without creating a new
2964 process. If *arguments* are specified, they shall be arguments to *command*. Redirection affects the
2965 current shell execution environment.2966 **OPTIONS**

2967 None.

2968 **OPERANDS**

2969 See the DESCRIPTION. |

2970 **STDIN**

2971 Not used. |

2972 **INPUT FILES**

2973 None.

2974 **ENVIRONMENT VARIABLES**

2975 None.

2976 **ASYNCHRONOUS EVENTS**

2977 Default. |

2978 **STDOUT**

2979 Not used. |

2980 **STDERR**

2981 The standard error shall be used only for diagnostic messages. |

2982 **OUTPUT FILES**

2983 None.

2984 **EXTENDED DESCRIPTION**

2985 None.

2986 **EXIT STATUS**2987 If *command* is specified, *exec* shall not return to the shell; rather, the exit status of the process shall
2988 be the exit status of the program implementing *command*, which overlaid the shell. If *command* is
2989 not found, the exit status shall be 127. If *command* is found, but it is not an executable utility, the
2990 exit status shall be 126. If a redirection error occurs (see Section 2.8.1 (on page 46)), the shell shall
2991 exit with a value in the range 1–125. Otherwise, *exec* shall return a zero exit status.

2992 **CONSEQUENCES OF ERRORS**

2993 Default.

2994 **APPLICATION USAGE**

2995 None.

2996 **EXAMPLES**2997 Open *readfile* as file descriptor 3 for reading:2998 `exec 3< readfile`2999 Open *writefile* as file descriptor 4 for writing:3000 `exec 4> writefile`

3001 Make file descriptor 5 a copy of file descriptor 0:

3002 `exec 5<&0`

3003 Close file descriptor 3:

3004 `exec 3<&-`3005 Cat the file **maggie** by replacing the current shell with the *cat* utility:3006 `exec cat maggie`3007 **RATIONALE**

3008 Most historical implementations were not conformant in that:

3009 `foo=bar exec cmd`3010 did not pass **foo** to **cmd**.3011 **FUTURE DIRECTIONS**

3012 None.

3013 **SEE ALSO**

3014 Section 2.14 (on page 64)

3015 **CHANGE HISTORY**3016 **Issue 6**3017 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
3018 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
3019 behavior is intended.

3020 **NAME**

3021 exit — cause the shell to exit

3022 **SYNOPSIS**3023 exit [*n*]3024 **DESCRIPTION**

3025 The *exit* utility shall cause the shell to exit with the exit status specified by the unsigned decimal
 3026 integer *n*. If *n* is specified, but its value is not between 0 and 255 inclusively, the exit status is
 3027 undefined.

3028 A *trap* on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is
 3029 invoked in that *trap* itself, in which case the shell shall exit immediately.

3030 **OPTIONS**

3031 None.

3032 **OPERANDS**

3033 See the DESCRIPTION. |

3034 **STDIN**

3035 Not used. |

3036 **INPUT FILES**

3037 None.

3038 **ENVIRONMENT VARIABLES**

3039 None.

3040 **ASYNCHRONOUS EVENTS**

3041 Default. |

3042 **STDOUT**

3043 Not used. |

3044 **STDERR**

3045 The standard error shall be used only for diagnostic messages. |

3046 **OUTPUT FILES**

3047 None.

3048 **EXTENDED DESCRIPTION**

3049 None.

3050 **EXIT STATUS**

3051 The exit status shall be *n*, if specified. Otherwise, the value shall be the exit value of the last
 3052 command executed, or zero if no command was executed. When *exit* is executed in a *trap* action,
 3053 the last command is considered to be the command that executed immediately preceding the
 3054 *trap* action.

3055 **CONSEQUENCES OF ERRORS**

3056 Default. |

3057 APPLICATION USAGE

3058 None.

3059 EXAMPLES

3060 Exit with a *true* value:

3061 exit 0

3062 Exit with a *false* value:

3063 exit 1

3064 RATIONALE

3065 As explained in other sections, certain exit status values have been reserved for special uses and
3066 should be used by applications only for those purposes:

3067 126 A file to be executed was found, but it was not an executable utility.

3068 127 A utility to be executed was not found.

3069 >128 A command was interrupted by a signal.

3070 FUTURE DIRECTIONS

3071 None.

3072 SEE ALSO

3073 Section 2.14 (on page 64)

3074 CHANGE HISTORY**3075 Issue 6**

3076 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
3077 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
3078 behavior is intended.

3079 **NAME**

3080 export — set the export attribute for variables

3081 **SYNOPSIS**

3082 export name [=word] . . .

3083 export -p

3084 **DESCRIPTION**

3085 The shell shall give the *export* attribute to the variables corresponding to the specified *names*,
 3086 which shall cause them to be in the environment of subsequently executed commands. If the
 3087 name of a variable is followed by =*word*, then the value of that variable shall be set to *word*.

3088 The *export* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-2001,
 3089 Section 12.2, Utility Syntax Guidelines.

3090 When **-p** is specified, *export* shall write to the standard output the names and values of all
 3091 exported variables, in the following format:

3092 "export %s=%s\n", <name>, <value>

3093 if *name* is set, and:

3094 "export %s\n", <name>

3095 if *name* is unset.

3096 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 3097 reinput to the shell as commands that achieve the same exporting results, except:

- 3098 1. Read-only variables with values cannot be reset.
- 3099 2. Variables that were unset at the time they were output need not be reset to the unset state
 3100 if a value is assigned to the variable between the time the state was saved and the time at
 3101 which the saved output is reinput to the shell.

3102 When no arguments are given, the results are unspecified.

3103 **OPTIONS**

3104 See the DESCRIPTION.

3105 **OPERANDS**

3106 See the DESCRIPTION.

3107 **STDIN**

3108 Not used.

3109 **INPUT FILES**

3110 None.

3111 **ENVIRONMENT VARIABLES**

3112 None.

3113 **ASYNCHRONOUS EVENTS**

3114 Default.

3115 **STDOUT**

3116 See the DESCRIPTION.

3117 **STDERR**

3118 The standard error shall be used only for diagnostic messages.

3119 **OUTPUT FILES**

3120 None.

3121 **EXTENDED DESCRIPTION**

3122 None.

3123 **EXIT STATUS**

3124 Zero.

3125 **CONSEQUENCES OF ERRORS**

3126 Default.

3127 **APPLICATION USAGE**

3128 None.

3129 **EXAMPLES**

3130 Export *PWD* and *HOME* variables:

3131 `export PWD HOME`

3132 Set and export the *PATH* variable:

3133 `export PATH=/local/bin:$PATH`

3134 Save and restore all exported variables:

3135 `export -p > temp-file`

3136 `unset a lot of variables`

3137 `... processing`

3138 `. temp-file`

3139 **RATIONALE**

3140 Some historical shells use the no-argument case as the functional equivalent of what is required
 3141 here with **-p**. This feature was left unspecified because it is not historical practice in all shells,
 3142 and some scripts may rely on the now-unspecified results on their implementations. Attempts to
 3143 specify the **-p** output as the default case were unsuccessful in achieving consensus. The **-p**
 3144 option was added to allow portable access to the values that can be saved and then later restored
 3145 using; for example, a *dot* script.

3146 **FUTURE DIRECTIONS**

3147 None.

3148 **SEE ALSO**

3149 Section 2.14 (on page 64)

3150 **CHANGE HISTORY**3151 **Issue 6**

3152 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

3153 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 3154 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
 3155 behavior is intended.

3156 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/6 is applied, adding the following text to
 3157 the end of the first paragraph of the DESCRIPTION: “If the name of a variable is followed by
 3158 =word, then the value of that variable shall be set to word.”. The reason for this change is that the
 3159 SYNOPSIS for *export* includes:

3160 `export name [=word] . . .`

3161 but the meaning of the optional “=word” is never explained in the text.

3162 **NAME**

3163 readonly — set the readonly attribute for variables

3164 **SYNOPSIS**

3165 readonly name [=word] . . .

3166 readonly -p

3167 **DESCRIPTION**

3168 The variables whose *names* are specified shall be given the *readonly* attribute. The values of
 3169 variables with the *readonly* attribute cannot be changed by subsequent assignment, nor can those
 3170 variables be unset by the *unset* utility. If the name of a variable is followed by =*word*, then the
 3171 value of that variable shall be set to *word*.

3172 The *readonly* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-2001,
 3173 Section 12.2, Utility Syntax Guidelines.

3174 When **-p** is specified, *readonly* writes to the standard output the names and values of all read-
 3175 only variables, in the following format:

3176 "readonly %s=%s\n", <name>, <value>

3177 if *name* is set, and

3178 "readonly %s\n", <name>

3179 if *name* is unset.

3180 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 3181 reinput to the shell as commands that achieve the same value and *readonly* attribute-setting
 3182 results in a shell execution environment in which:

- 3183 1. Variables with values at the time they were output do not have the *readonly* attribute set.
- 3184 2. Variables that were unset at the time they were output do not have a value at the time at
 3185 which the saved output is reinput to the shell.

3186 When no arguments are given, the results are unspecified.

3187 **OPTIONS**

3188 See the DESCRIPTION.

3189 **OPERANDS**

3190 See the DESCRIPTION.

3191 **STDIN**

3192 Not used.

3193 **INPUT FILES**

3194 None.

3195 **ENVIRONMENT VARIABLES**

3196 None.

3197 **ASYNCHRONOUS EVENTS**

3198 Default.

3199 **STDOUT**

3200 See the DESCRIPTION.

3201 STDERR

3202 The standard error shall be used only for diagnostic messages.

3203 OUTPUT FILES

3204 None.

3205 EXTENDED DESCRIPTION

3206 None.

3207 EXIT STATUS

3208 Zero.

3209 CONSEQUENCES OF ERRORS

3210 Default.

3211 APPLICATION USAGE

3212 None.

3213 EXAMPLES

3214 `readonly HOME PWD`

3215 RATIONALE

3216 Some historical shells preserve the *readonly* attribute across separate invocations. This volume of
3217 IEEE Std 1003.1-2001 allows this behavior, but does not require it.

3218 The `-p` option allows portable access to the values that can be saved and then later restored
3219 using, for example, a *dot* script. Also see the RATIONALE for *export* (on page 79) for a
3220 description of the no-argument and `-p` output cases and a related example.

3221 Read-only functions were considered, but they were omitted as not being historical practice or
3222 particularly useful. Furthermore, functions must not be read-only across invocations to preclude
3223 “spoofing” (spoofing is the term for the practice of creating a program that acts like a well-
3224 known utility with the intent of subverting the real intent of the user) of administrative or
3225 security-relevant (or security-conscious) shell scripts.

3226 FUTURE DIRECTIONS

3227 None.

3228 SEE ALSO

3229 Section 2.14 (on page 64)

3230 CHANGE HISTORY**3231 Issue 6**

3232 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

3233 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
3234 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
3235 behavior is intended.

3236 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/7 is applied, adding the following text to
3237 the end of the first paragraph of the DESCRIPTION: “If the name of a variable is followed by
3238 `=word`, then the value of that variable shall be set to *word*.”. The reason for this change is that the
3239 SYNOPSIS for *readonly* includes:

3240 `readonly name [=word] . . .`

3241 but the meaning of the optional “`=word`” is never explained in the text.

3242 **NAME**

3243 return — return from a function

3244 **SYNOPSIS**3245 return [*n*]3246 **DESCRIPTION**

3247 The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the
 3248 shell is not currently executing a function or *dot* script, the results are unspecified.

3249 **OPTIONS**

3250 None.

3251 **OPERANDS**

3252 See the DESCRIPTION.

3253 **STDIN**

3254 Not used.

3255 **INPUT FILES**

3256 None.

3257 **ENVIRONMENT VARIABLES**

3258 None.

3259 **ASYNCHRONOUS EVENTS**

3260 Default.

3261 **STDOUT**

3262 Not used.

3263 **STDERR**

3264 The standard error shall be used only for diagnostic messages.

3265 **OUTPUT FILES**

3266 None.

3267 **EXTENDED DESCRIPTION**

3268 None.

3269 **EXIT STATUS**

3270 The value of the special parameter '*?*' shall be set to *n*, an unsigned decimal integer, or to the
 3271 exit status of the last command executed if *n* is not specified. If the value of *n* is greater than 255,
 3272 the results are undefined. When *return* is executed in a *trap* action, the last command is
 3273 considered to be the command that executed immediately preceding the *trap* action.

3274 **CONSEQUENCES OF ERRORS**

3275 Default.

3276 **APPLICATION USAGE**

3277 None.

3278 **EXAMPLES**

3279 None.

3280 **RATIONALE**

3281 The behavior of *return* when not in a function or *dot* script differs between the System V shell
 3282 and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is
 3283 the same as *exit*.

3284 The results of returning a number greater than 255 are undefined because of differing practices
3285 in the various historical implementations. Some shells AND out all but the low-order 8 bits;
3286 others allow larger values, but not of unlimited size.

3287 See the discussion of appropriate exit status values under *exit* (on page 77).

3288 **FUTURE DIRECTIONS**

3289 None.

3290 **SEE ALSO**

3291 Section 2.14 (on page 64)

3292 **CHANGE HISTORY**

3293 **Issue 6**

3294 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
3295 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
3296 behavior is intended.

3297 **NAME**

3298 set — set or unset options and positional parameters

3299 **SYNOPSIS**3300 xSI set [-abCefmnuvx] [-h] [-o *option*] [*argument...*]3301 xSI set [+abCefmnuvx] [+h] [+o *option*] [*argument...*]3302 set -- [*argument...*]

3303 set -o

3304 set +o

3305 **DESCRIPTION**

3306 If no *options* or *arguments* are specified, *set* shall write the names and values of all shell variables
 3307 in the collation sequence of the current locale. Each *name* shall start on a separate line, using the
 3308 format:

3309 "%s=%s\n", <name>, <value>

3310 The *value* string shall be written with appropriate quoting; see the description of shell quoting in
 3311 Section 2.2 (on page 30). The output shall be suitable for reinput to the shell, setting or resetting,
 3312 as far as possible, the variables that are currently set; read-only variables cannot be reset.

3313 When options are specified, they shall set or unset attributes of the shell, as described below.
 3314 When *arguments* are specified, they cause positional parameters to be set or unset, as described
 3315 below. Setting or unsetting attributes and positional parameters are not necessarily related
 3316 actions, but they can be combined in a single invocation of *set*.

3317 The *set* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-2001,
 3318 Section 12.2, Utility Syntax Guidelines except that options can be specified with either a leading
 3319 hyphen (meaning enable the option) or plus sign (meaning disable it) unless otherwise specified.

3320 Implementations shall support the options in the following list in both their hyphen and plus-
 3321 sign forms. These options can also be specified as options to *sh*.

3322 **-a** When this option is on, the *export* attribute shall be set for each variable to which an
 3323 assignment is performed; see the Base Definitions volume of IEEE Std 1003.1-2001, Section
 3324 4.21, Variable Assignment. If the assignment precedes a utility name in a command, the
 3325 *export* attribute shall not persist in the current execution environment after the utility
 3326 completes, with the exception that preceding one of the special built-in utilities causes the
 3327 *export* attribute to persist after the built-in has completed. If the assignment does not
 3328 precede a utility name in the command, or if the assignment is a result of the operation of
 3329 the *getopts* or *read* utilities, the *export* attribute shall persist until the variable is unset.

3330 **-b** This option shall be supported if the implementation supports the User Portability Utilities
 3331 option. It shall cause the shell to notify the user asynchronously of background job
 3332 completions. The following message is written to standard error:

3333 "[%d] %c %s%s\n", <job-number>, <current>, <status>, <job-name>

3334 where the fields shall be as follows:

3335 <current> The character '+' identifies the job that would be used as a default for
 3336 the *fg* or *bg* utilities; this job can also be specified using the *job_id* "%+" or
 3337 "%%". The character '-' identifies the job that would become the default
 3338 if the current default job were to exit; this job can also be specified using
 3339 the *job_id* "%-". For other jobs, this field is a <space>. At most one job
 3340 can be identified with '+' and at most one job can be identified with '-'.

3341		If there is any suspended job, then the current job shall be a suspended
3342		job. If there are at least two suspended jobs, then the previous job also
3343		shall be a suspended job.
3344	<job-number>	A number that can be used to identify the process group to the <i>wait</i> , <i>fg</i> , <i>bg</i> ,
3345		and <i>kill</i> utilities. Using these utilities, the job can be identified by
3346		prefixing the job number with ' <i>%</i> '.
3347	<status>	Unspecified.
3348	<job-name>	Unspecified.
3349		When the shell notifies the user a job has been completed, it may remove the job's process
3350		ID from the list of those known in the current shell execution environment; see Section
3351		2.9.3.1 (on page 50). Asynchronous notification shall not be enabled by default.
3352	-C	(Uppercase C.) Prevent existing files from being overwritten by the shell's '>' redirection
3353		operator (see Section 2.7.2 (on page 44)); the "> " redirection operator shall override this
3354		<i>noclobber</i> option for an individual file.
3355	-e	When this option is on, if a simple command fails for any of the reasons listed in Section
3356		2.8.1 (on page 46) or returns an exit status value >0, and is not part of the compound list
3357		following a while , until , or if keyword, and is not a part of an AND or OR list, and is not a
3358		pipeline preceded by the ! reserved word, then the shell shall immediately exit.
3359	-f	The shell shall disable pathname expansion.
3360	XSI -h	Locate and remember utilities invoked by functions as those functions are defined (the
3361		utilities are normally located when the function is executed).
3362	-m	This option shall be supported if the implementation supports the User Portability Utilities
3363		option. All jobs shall be run in their own process groups. Immediately before the shell issues
3364		a prompt after completion of the background job, a message reporting the exit status of the
3365		background job shall be written to standard error. If a foreground job stops, the shell shall
3366		write a message to standard error to that effect, formatted as described by the <i>jobs</i> utility. In
3367		addition, if a job changes status other than exiting (for example, if it stops for input or
3368		output or is stopped by a SIGSTOP signal), the shell shall write a similar message
3369		immediately prior to writing the next prompt. This option is enabled by default for
3370		interactive shells.
3371	-n	The shell shall read commands but does not execute them; this can be used to check for
3372		shell script syntax errors. An interactive shell may ignore this option.
3373	-o	Write the current settings of the options to standard output in an unspecified format.
3374	+o	Write the current option settings to standard output in a format that is suitable for reinput
3375		to the shell as commands that achieve the same options settings.
3376	-o option	
3377		This option is supported if the system supports the User Portability Utilities option. It shall
3378		set various options, many of which shall be equivalent to the single option letters. The
3379		following values of <i>option</i> shall be supported:
3380	<i>allexport</i>	Equivalent to -a .
3381	<i>errexit</i>	Equivalent to -e .
3382	<i>ignoreeof</i>	Prevent an interactive shell from exiting on end-of-file. This setting prevents
3383		accidental logouts when <control>-D is entered. A user shall explicitly <i>exit</i> to
3384		leave the interactive shell.

3385	<i>monitor</i>	Equivalent to -m . This option is supported if the system supports the User Portability Utilities option.
3386		
3387	<i>noclobber</i>	Equivalent to -C (uppercase C).
3388	<i>noglob</i>	Equivalent to -f .
3389	<i>noexec</i>	Equivalent to -n .
3390	<i>nolog</i>	Prevent the entry of function definitions into the command history; see Command History List (on page 854).
3391		
3392	<i>notify</i>	Equivalent to -b .
3393	<i>nounset</i>	Equivalent to -u .
3394	<i>verbose</i>	Equivalent to -v .
3395	<i>vi</i>	Allow shell command line editing using the built-in <i>vi</i> editor. Enabling <i>vi</i> mode shall disable any other command line editing mode provided as an implementation extension.
3396		
3397		
3398		It need not be possible to set <i>vi</i> mode on for certain block-mode terminals.
3399	<i>xtrace</i>	Equivalent to -x .
3400	-u	The shell shall write a message to standard error when it tries to expand a variable that is not set and immediately exit. An interactive shell shall not exit.
3401		
3402	-v	The shell shall write its input to standard error as it is read.
3403	-x	The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.
3404		
3405		
3406		The default for all these options shall be off (unset) unless stated otherwise in the description of the option or unless the shell was invoked with them on; see <i>sh</i> .
3407		
3408		The remaining arguments shall be assigned in order to the positional parameters. The special parameter ' # ' shall be set to reflect the number of positional parameters. All positional parameters shall be unset before any new values are assigned.
3409		
3410		
3411		The special argument " -- " immediately following the <i>set</i> command name can be used to delimit the arguments if the first argument begins with ' + ' or ' - ', or to prevent inadvertent listing of all shell variables when there are no arguments. The command <i>set --</i> without <i>argument</i> shall unset all positional parameters and set the special parameter ' # ' to zero.
3412		
3413		
3414		
3415	OPTIONS	
3416		See the DESCRIPTION.
3417	OPERANDS	
3418		See the DESCRIPTION.
3419	STDIN	
3420		Not used.
3421	INPUT FILES	
3422		None.

3423 **ENVIRONMENT VARIABLES**

3424 None.

3425 **ASYNCHRONOUS EVENTS**

3426 Default.

3427 **STDOUT**

3428 See the DESCRIPTION.

3429 **STDERR**

3430 The standard error shall be used only for diagnostic messages.

3431 **OUTPUT FILES**

3432 None.

3433 **EXTENDED DESCRIPTION**

3434 None.

3435 **EXIT STATUS**

3436 Zero.

3437 **CONSEQUENCES OF ERRORS**

3438 Default.

3439 **APPLICATION USAGE**

3440 None.

3441 **EXAMPLES**

3442 Write out all variables and their values:

3443 `set`

3444 Set \$1, \$2, and \$3 and set "\$#" to 3:

3445 `set c a b`3446 Turn on the `-x` and `-v` options:3447 `set -xv`

3448 Unset all positional parameters:

3449 `set --`3450 Set \$1 to the value of `x`, even if it begins with `'-'` or `'+'`:3451 `set -- "$x"`3452 Set the positional parameters to the expansion of `x`, even if `x` expands with a leading `'-'` or `'+'`:3453 `set -- $x`3454 **RATIONALE**

3455 The `set --` form is listed specifically in the SYNOPSIS even though this usage is implied by the
 3456 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether
 3457 the `set --` form might be misinterpreted as being equivalent to `set` without any options or
 3458 arguments. The functionality of this form has been adopted from the KornShell. In System V, `set`
 3459 `--` only unsets parameters if there is at least one argument; the only way to unset all parameters
 3460 is to use `shift`. Using the KornShell version should not affect System V scripts because there
 3461 should be no reason to issue it without arguments deliberately; if it were issued as, for example:

3462 `set -- "$@"`

and there were in fact no arguments resulting from "\$@", unsetting the parameters would have no result.

The `set +` form in early proposals was omitted as being an unnecessary duplication of `set` alone and not widespread historical practice.

The `noclobber` option was changed to allow `set -C` as well as the `set -o noclobber` option. The single-letter version was added so that the historical "\$-" paradigm would not be broken; see Section 2.5.2 (on page 34).

The `-h` flag is related to command name hashing and is only required on XSI-conformant systems.

The following `set` flags were omitted intentionally with the following rationale:

-k The `-k` flag was originally added by the author of the Bourne shell to make it easier for users of pre-release versions of the shell. In early versions of the Bourne shell the construct `set name=value` had to be used to assign values to shell variables. The problem with `-k` is that the behavior affects parsing, virtually precluding writing any compilers. To explain the behavior of `-k`, it is necessary to describe the parsing algorithm, which is implementation-defined. For example:

```
set -k; echo name=value
```

and:

```
set -k
echo name=value
```

behave differently. The interaction with functions is even more complex. What is more, the `-k` flag is never needed, since the command line could have been reordered.

-t The `-t` flag is hard to specify and almost never used. The only known use could be done with here-documents. Moreover, the behavior with `ksh` and `sh` differs. The reference page says that it exits after reading and executing one command. What is one command? If the input is `date;date`, `sh` executes both `date` commands while `ksh` does only the first.

Consideration was given to rewriting `set` to simplify its confusing syntax. A specific suggestion was that the `unset` utility should be used to unset options instead of using the non-`getopt()`-able `+option` syntax. However, the conclusion was reached that the historical practice of using `+option` was satisfactory and that there was no compelling reason to modify such widespread historical practice.

The `-o` option was adopted from the KornShell to address user needs. In addition to its generally friendly interface, `-o` is needed to provide the `vi` command line editing mode, for which historical practice yields no single-letter option name. (Although it might have been possible to invent such a letter, it was recognized that other editing modes would be developed and `-o` provides ample name space for describing such extensions.)

Historical implementations are inconsistent in the format used for `-o` option status reporting. The `+o` format without an option-argument was added to allow portable access to the options that can be saved and then later restored using, for instance, a dot script.

Historically, `sh` did trace the command `set +x`, but `ksh` did not.

The `ignoreeof` setting prevents accidental logouts when the end-of-file character (typically `<control>-D`) is entered. A user shall explicitly `exit` to leave the interactive shell.

The `set -m` option was added to apply only to the UPE because it applies primarily to interactive use, not shell script applications.

3507 The ability to do asynchronous notification became available in the 1988 version of the
 3508 KornShell. To have it occur, the user had to issue the command:

3509 `trap "jobs -n" CLD`

3510 The C shell provides two different levels of an asynchronous notification capability. The
 3511 environment variable *notify* is analogous to what is done in *set -b* or *set -o notify*. When set, it
 3512 notifies the user immediately of background job completions. When unset, this capability is
 3513 turned off.

3514 The other notification ability comes through the built-in utility *notify*. The syntax is:

3515 `notify [%job ... ]`

3516 By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when
 3517 the state of the current job changes. If given operands, *notify* asynchronously informs the user of
 3518 changes in the states of the specified jobs.

3519 To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*,
 3520 nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX
 3521 environment variable name).

3522 The *set -b* option was selected as a compromise.

3523 The *notify* built-in was considered to have more functionality than was required for simple
 3524 asynchronous notification.

3525 **FUTURE DIRECTIONS**

3526 None.

3527 **SEE ALSO**

3528 Section 2.14 (on page 64)

3529 **CHANGE HISTORY**

3530 **Issue 6**

3531 The obsolescent *set* command name followed by *'-'* has been removed.

3532 The following new requirements on POSIX implementations derive from alignment with the
 3533 Single UNIX Specification:

- 3534 • The *nolog* option is added to *set -o*.

3535 IEEE PASC Interpretation 1003.2 #167 is applied, clarifying that the options default also takes
 3536 into account the description of the option.

3537 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 3538 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
 3539 behavior is intended.

3540 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/8 is applied, changing the square brackets
 3541 in the example in RATIONALE to be in bold, which is the typeface used for optional items.

3542 NAME

3543 shift — shift positional parameters

3544 SYNOPSIS

3545 shift [n]

3546 DESCRIPTION

3547 The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of
 3548 parameter (1+n), parameter 2 shall be assigned the value of parameter (2+n), and so on. The
 3549 parameters represented by the numbers "\$#" down to "\$#-n+1" shall be unset, and the
 3550 parameter '#' is updated to reflect the new number of positional parameters.

3551 The value n shall be an unsigned decimal integer less than or equal to the value of the special
 3552 parameter '#'. If n is not given, it shall be assumed to be 1. If n is 0, the positional and special
 3553 parameters are not changed.

3554 OPTIONS

3555 None.

3556 OPERANDS

3557 See the DESCRIPTION.

3558 STDIN

3559 Not used.

3560 INPUT FILES

3561 None.

3562 ENVIRONMENT VARIABLES

3563 None.

3564 ASYNCHRONOUS EVENTS

3565 Default.

3566 STDOUT

3567 Not used.

3568 STDERR

3569 The standard error shall be used only for diagnostic messages.

3570 OUTPUT FILES

3571 None.

3572 EXTENDED DESCRIPTION

3573 None.

3574 EXIT STATUS

3575 The exit status is >0 if n>\$#; otherwise, it is zero.

3576 CONSEQUENCES OF ERRORS

3577 Default.

3578 **APPLICATION USAGE**

3579 None.

3580 **EXAMPLES**

3581 \$ set a b c d e

3582 \$ shift 2

3583 \$ echo \$*

3584 c d e

3585 **RATIONALE**

3586 None.

3587 **FUTURE DIRECTIONS**

3588 None.

3589 **SEE ALSO**

3590 Section 2.14 (on page 64)

3591 **CHANGE HISTORY**3592 **Issue 6**

3593 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
3594 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
3595 behavior is intended.

3596 **NAME**

3597 times — write process times

3598 **SYNOPSIS**

3599 times

3600 **DESCRIPTION**

3601 The *times* utility shall write the accumulated user and system times for the shell and for all of its
 3602 child processes, in the following POSIX locale format:

```
3603       "%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,
3604       <shell user seconds>, <shell system minutes>,
3605       <shell system seconds>, <children user minutes>,
3606       <children user seconds>, <children system minutes>,
3607       <children system seconds>
```

3608 The four pairs of times shall correspond to the members of the **<sys/times.h> tms** structure
 3609 (defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) as
 3610 returned by *times()*: *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime*, respectively.

3611 **OPTIONS**

3612 None.

3613 **OPERANDS**

3614 None.

3615 **STDIN**

3616 Not used.

3617 **INPUT FILES**

3618 None.

3619 **ENVIRONMENT VARIABLES**

3620 None.

3621 **ASYNCHRONOUS EVENTS**

3622 Default.

3623 **STDOUT**

3624 See the DESCRIPTION.

3625 **STDERR**

3626 The standard error shall be used only for diagnostic messages.

3627 **OUTPUT FILES**

3628 None.

3629 **EXTENDED DESCRIPTION**

3630 None.

3631 **EXIT STATUS**

3632 Zero.

3633 **CONSEQUENCES OF ERRORS**

3634 Default.

3635 **APPLICATION USAGE**

3636 None.

3637 **EXAMPLES**

3638 \$ times

3639 0m0.43s 0m1.11s

3640 8m44.18s 1m43.23s

3641 **RATIONALE**3642 The *times* special built-in from the Single UNIX Specification is now required for all conforming
3643 shells.3644 **FUTURE DIRECTIONS**

3645 None.

3646 **SEE ALSO**

3647 Section 2.14 (on page 64)

3648 **CHANGE HISTORY**3649 **Issue 6**3650 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/9 is applied, changing text in the
3651 DESCRIPTION from: “Write the accumulated user and system times for the shell and for all of
3652 its child processes ...” to: “The *times* utility shall write the accumulated user and system times
3653 for the shell and for all of its child processes ...”.

3654 **NAME**

3655 trap — trap signals

3656 **SYNOPSIS**3657 trap [*action condition ...*]3658 **DESCRIPTION**

3659 If *action* is '-', the shell shall reset each *condition* to the default value. If *action* is null (""), the
 3660 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read
 3661 and executed by the shell when one of the corresponding conditions arises. The action of *trap*
 3662 shall override a previous action (either default action or one explicitly set). The value of "\$?"
 3663 after the *trap* action completes shall be the value it had before *trap* was invoked.

3664 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,
 3665 without the SIG prefix, as listed in the tables of signal names in the <signal.h> header defined in
 3666 the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers; for example, HUP,
 3667 INT, QUIT, TERM. Implementations may permit names with the SIG prefix or ignore case in
 3668 signal names as an extension. Setting a trap for SIGKILL or SIGSTOP produces undefined
 3669 results.

3670 The environment in which the shell executes a *trap* on EXIT shall be identical to the environment
 3671 immediately after the last command executed before the *trap* on EXIT was taken.

3672 Each time *trap* is invoked, the *action* argument shall be processed in a manner equivalent to:

3673 eval *action*

3674 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although
 3675 no error need be reported when attempting to do so. An interactive shell may reset or catch
 3676 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed
 3677 with another *trap* command.

3678 When a subshell is entered, traps that are not being ignored are set to the default actions. This
 3679 does not imply that the *trap* command cannot be used within the subshell to set new traps.

3680 The *trap* command with no arguments shall write to standard output a list of commands
 3681 associated with each condition. The format shall be:

3682 "trap -- %s %s ...\n", <action>, <condition> ...

3683 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 3684 reinput to the shell as commands that achieve the same trapping results. For example:

```
3685 save_traps=$(trap)
3686 ...
3687 eval "$save_traps"
```

3688 XSI the following signal names:

3690 1 SIGHUP

3691 2 SIGINT

3692 3 SIGQUIT

3693 6 SIGABRT

3694 9 SIGKILL

3695 14 SIGALRM

3696	15	SIGTERM	
3697			
3698		The <i>trap</i> special built-in shall conform to the Base Definitions volume of IEEE Std 1003.1-2001,	
3699		Section 12.2, Utility Syntax Guidelines.	
3700		OPTIONS	
3701		None.	
3702		OPERANDS	
3703		See the DESCRIPTION.	
3704		STDIN	
3705		Not used.	
3706		INPUT FILES	
3707		None.	
3708		ENVIRONMENT VARIABLES	
3709		None.	
3710		ASYNCHRONOUS EVENTS	
3711		Default.	
3712		STDOUT	
3713		See the DESCRIPTION.	
3714		STDERR	
3715		The standard error shall be used only for diagnostic messages.	
3716		OUTPUT FILES	
3717		None.	
3718		EXTENDED DESCRIPTION	
3719		None.	
3720		EXIT STATUS	
3721	XSI	If the trap name or number is invalid, a non-zero exit status shall be returned; otherwise, zero	
3722	XSI	shall be returned. For both interactive and non-interactive shells, invalid signal names or	
3723		numbers shall not be considered a syntax error and do not cause the shell to abort.	
3724		CONSEQUENCES OF ERRORS	
3725		Default.	
3726		APPLICATION USAGE	
3727		None.	
3728		EXAMPLES	
3729		Write out a list of all traps and actions:	
3730		trap	
3731		Set a trap so the <i>logout</i> utility in the directory referred to by the <i>HOME</i> environment variable	
3732		executes when the shell terminates:	
3733		trap '\$HOME/logout' EXIT	
3734		or:	
3735		trap '\$HOME/logout' 0	
3736		Unset traps on INT, QUIT, TERM, and EXIT:	

3737 trap - INT QUIT TERM EXIT

3738 **RATIONALE**

3739 Implementations may permit lowercase signal names as an extension. Implementations may
3740 also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill*
3741 utilities in this volume of IEEE Std 1003.1-2001 are now consistent in their omission of the SIG
3742 prefix for signal names. Some *kill* implementations do not allow the prefix, and *kill -l* lists the
3743 signals without prefixes.

3744 Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but
3745 it has no effect. Portable POSIX applications cannot attempt to trap these signals.

3746 The output format is not historical practice. Since the output of historical *trap* commands is not
3747 portable (because numeric signal values are not portable) and had to change to become so, an
3748 opportunity was taken to format the output in a way that a shell script could use to save and
3749 then later reuse a trap if it wanted.

3750 The KornShell uses an **ERR** trap that is triggered whenever *set -e* would cause an exit. This is
3751 allowable as an extension, but was not mandated, as other shells have not used it.

3752 The text about the environment for the EXIT trap invalidates the behavior of some historical
3753 versions of interactive shells which, for example, close the standard input before executing a
3754 trap on 0. For example, in some historical interactive shell sessions the following trap on 0 would
3755 always print "--":

3756 trap 'read foo; echo "--\$foo--"' 0

3757 **FUTURE DIRECTIONS**

3758 None.

3759 **SEE ALSO**

3760 Section 2.14 (on page 64)

3761 **CHANGE HISTORY**

3762 **Issue 6**

3763 XSI-conforming implementations provide the mapping of signal names to numbers given above
3764 (previously this had been marked obsolescent). Other implementations need not provide this
3765 optional mapping.

3766 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
3767 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
3768 behavior is intended.

3769 **NAME**

3770 unset — unset values and attributes of variables and functions

3771 **SYNOPSIS**3772 unset [-fv] *name* ...3773 **DESCRIPTION**3774 Each variable or function specified by *name* shall be unset.3775 If **-v** is specified, *name* refers to a variable name and the shell shall unset it and remove it from
3776 the environment. Read-only variables cannot be unset.3777 If **-f** is specified, *name* refers to a function and the shell shall unset the function definition.3778 If neither **-f** nor **-v** is specified, *name* refers to a variable; if a variable by that name does not
3779 exist, it is unspecified whether a function by that name, if any, shall be unset.3780 Unsetting a variable or function that was not previously set shall not be considered an error and
3781 does not cause the shell to abort.3782 The *unset* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-2001,
3783 Section 12.2, Utility Syntax Guidelines.

3784 Note that:

3785 VARIABLE=

3786 is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to ". Also, the
3787 variables that can be *unset* should not be misinterpreted to include the special parameters (see
3788 Section 2.5.2 (on page 34)).3789 **OPTIONS**

3790 See the DESCRIPTION. |

3791 **OPERANDS**

3792 See the DESCRIPTION. |

3793 **STDIN**

3794 Not used. |

3795 **INPUT FILES**

3796 None.

3797 **ENVIRONMENT VARIABLES**

3798 None.

3799 **ASYNCHRONOUS EVENTS**

3800 Default. |

3801 **STDOUT**

3802 Not used. |

3803 **STDERR**

3804 The standard error shall be used only for diagnostic messages. |

3805 **OUTPUT FILES**

3806 None.

3807 **EXTENDED DESCRIPTION**

3808 None.

3809 **EXIT STATUS**

- 3810 0 All *name* operands were successfully unset.
3811 >0 At least one *name* could not be unset.

3812 **CONSEQUENCES OF ERRORS**

- 3813 Default.

3814 **APPLICATION USAGE**

- 3815 None.

3816 **EXAMPLES**

- 3817 Unset *VISUAL* variable:
3818 unset -v *VISUAL*
3819 Unset the functions **foo** and **bar**:
3820 unset -f foo bar

3821 **RATIONALE**

- 3822 Consideration was given to omitting the **-f** option in favor of an *unfunction* utility, but the
3823 standard developers decided to retain historical practice.
3824 The **-v** option was introduced because System V historically used one name space for both
3825 variables and functions. When *unset* is used without options, System V historically unset either a
3826 function or a variable, and there was no confusion about which one was intended. A portable
3827 POSIX application can use *unset* without an option to unset a variable, but not a function; the **-f**
3828 option must be used.

3829 **FUTURE DIRECTIONS**

- 3830 None.

3831 **SEE ALSO**

- 3832 Section 2.14 (on page 64)

3833 **CHANGE HISTORY**3834 **Issue 6**

- 3835 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
3836 sections use terms as described in the Utility Description Defaults (Section 1.11). No change in
3837 behavior is intended.

Batch Environment Services

This chapter describes the services and utilities that shall be implemented on all systems that claim conformance to the Batch Environment Services and Utilities option. This functionality is dependent on support of this option (and the rest of this section is not further shaded for this option).

3.1 General Concepts

3.1.1 Batch Client-Server Interaction

Batch jobs are created and managed by batch servers. A batch client interacts with a batch server to access batch services on behalf of the user. In order to use batch services, a user must have access to a batch client.

A batch server is a computational entity, such as a daemon process, that provides batch services. Batch servers route, queue, modify, and execute batch jobs on behalf of batch clients.

The batch utilities described in this volume of IEEE Std 1003.1-2001 (and listed in Table 3-1) are clients of batch services; they allow users to perform actions on the job such as creating, modifying, and deleting batch jobs from a shell command line. Although these batch utilities may be said to accomplish certain services, they actually obtain services on behalf of a user by means of requests to batch servers.

Table 3-1 Batch Utilities

<i>qalter</i>	<i>qmove</i>	<i>qrls</i>	<i>qstat</i>
<i>qdel</i>	<i>qmsg</i>	<i>qselect</i>	<i>qsub</i>
<i>qhold</i>	<i>qrerun</i>	<i>qsig</i>	

Client-server interaction takes place by means of the batch requests defined in this chapter. Because direct access to batch jobs and queues is limited to batch servers, clients and servers of different implementations can interoperate, since dependencies on private structures for batch jobs and queues are limited to batch servers. Also, batch servers may be clients of other batch servers.

3.1.2 Batch Queues

Two types of batch queue are described: routing queues and execution queues. When a batch job is placed in a routing queue, it is a candidate for routing. A batch job is removed from routing queues under the following conditions:

- The batch job has been routed to another queue.
- The batch job has been deleted from the batch queue.
- The batch job has been aborted.

When a batch job is placed in an execution queue, it is a candidate for execution.

A batch job is removed from an execution queue under the following conditions:

- 3873 • The batch job has been executed and exited.
- 3874 • The batch job has been aborted.
- 3875 • The batch job has been deleted from the batch queue.
- 3876 • The batch job has been moved to another queue.

3877 Access to a batch queue is limited to the batch server that manages the batch queue. Clients
 3878 never access a batch queue or a batch job directly, either to read or write information; all client
 3879 access to batch queues or jobs takes place through batch servers.

3880 **3.1.3 Batch Job Creation**

3881 When a batch server creates a batch job on behalf of a client, it shall assign a batch job identifier
 3882 to the job. A batch job identifier consists of both a sequence number that is unique among the
 3883 sequence numbers issued by that server and the name of the server. Since the batch server name
 3884 is unique within a name space, the job identifier is likewise unique within the name space.

3885 The batch server that creates a batch job shall return the batch server-assigned job identifier to
 3886 the client that requested the job creation. If the batch server routes or moves the job to another
 3887 server, it sends the job identifier with the job. Once assigned, the job identifier of a batch job shall
 3888 never change.

3889 **3.1.4 Batch Job Tracking**

3890 Since a batch job may be moved after creation, the batch server name component of the job
 3891 identifier need not indicate the location of the job. An implementation may provide a batch job
 3892 tracking mechanism, in which case the user generally does not need to know the location of the
 3893 job. However, an implementation need not provide a batch job tracking mechanism, in which
 3894 case the user must find routed jobs by probing the possible destinations.

3895 **3.1.5 Batch Job Routing**

3896 To route a batch job, a batch server either moves the job to some other queue that is managed by
 3897 the batch server, or requests that some other batch server accept the job.

3898 Each routing queue has one or more queues to which it can route batch jobs. The batch server
 3899 administrator creates routing queues.

3900 A batch server may route a batch job from a routing queue to another routing queue. Batch
 3901 servers shall prevent or otherwise handle cases of circular routing paths. As a deferred service, a
 3902 batch server routes jobs from the routing queues that it manages. The algorithm by which a
 3903 batch server selects a batch queue to which to route a batch job is implementation-defined.

3904 A batch job need not be eligible for routing to all the batch queues fed by the routing queue from
 3905 which it is routed. A batch server that has been asked to accept the job may reject the request if
 3906 the job requires resources that are unavailable to that batch server, or if the client is not
 3907 authorized to access the batch server.

3908 Batch servers may route high-priority jobs before low-priority jobs, but, on other than
 3909 overloaded systems, the effect may be imperceptible to the user. If all the batch servers fed by a
 3910 routing queue reject requests to accept the job for reasons that are permanent, the batch server
 3911 that manages the job shall abort the job. If all or some rejections are temporary, the batch server
 3912 should try to route the job again at some later point.

3913 The reasons for rejecting a batch job are implementation-defined.

3914 The reasons for which the routing should be retried later and the reasons for which the job
3915 should be aborted are also implementation-defined.

3916 **3.1.6 Batch Job Execution**

3917 To execute a batch job is to create a session leader (a process) that runs the shell program
3918 indicated by the *Shell_Path* attribute of the job. The script shall be passed to the program as its
3919 standard input. An implementation may pass the script to the program by other
3920 implementation-defined means. At the time a batch job begins execution, it is defined to enter
3921 the RUNNING state. The primary program that is executed by a batch job is typically, though
3922 not necessarily, a shell program.

3923 A batch server shall execute eligible jobs as a deferred service—no client request is necessary
3924 once the batch job is created and eligible. However, the attributes of a batch job, such as the job
3925 hold type, may render the job ineligible. A batch server shall scan the execution queues that it
3926 manages for jobs that are eligible for execution. The algorithm by which the batch server selects
3927 eligible jobs for execution is implementation-defined.

3928 As part of creating the process for the batch job, the batch server shall open the standard output
3929 and standard error streams of the session.

3930 The attributes of a batch job may indicate that the batch server executing the job shall send mail
3931 to a list of users at the time it begins execution of the job.

3932 **3.1.7 Batch Job Exit**

3933 When the session leader of an executing job terminates, the job exits. As part of exiting a batch
3934 job, the batch server that manages the job shall remove the job from the batch queue in which it
3935 resides. The server shall transfer output files of the job to a location described by the attributes of
3936 the job.

3937 The attributes of a batch job may indicate that the batch server managing the job shall send mail
3938 to a list of users at the time the job exits.

3939 **3.1.8 Batch Job Abort**

3940 A batch server shall abort jobs for which a required deferred service cannot be performed. The
3941 attributes of a batch job may indicate that the batch server that aborts the job shall send mail to a
3942 list of users at the time it aborts the job.

3943 **3.1.9 Batch Authorization**

3944 Clients, such as the batch environment utilities (marked BE), access batch services by means of
3945 requests to one or more batch servers. To acquire the services of any given batch server, the user
3946 identifier under which the client runs must be authorized to use that batch server.

3947 The user with an associated user name that creates a batch job shall own the job and can perform
3948 actions such as read, modify, delete, and move.

3949 A user identifier of the same value at a different host need not be the same user. For example,
3950 user name *smith* at host **alpha** may or may not represent the same person as user name *smith* at
3951 host **beta**. Likewise, the same person may have access to different user names on different hosts.

3952 An implementation may optionally provide an authorization mechanism that permits one user
3953 name to access jobs under another user name.

3954 A process on a client host may be authorized to run processes under multiple user names at a
3955 batch server host. Where appropriate, the utilities defined in this volume of IEEE Std 1003.1-2001

3956 provide a means for a user to choose from among such user names when creating or modifying a
 3957 batch job.

3958 **3.1.10 Batch Administration**

3959 The processing of a batch job by a batch server is affected by the attributes of the job. The
 3960 processing of a batch job may also be affected by the attributes of the batch queue in which the
 3961 job resides and by the status of the batch server that manages the job. See also the Base
 3962 Definitions volume of IEEE Std 1003.1-2001, Chapter 3, Definitions for batch definitions.

3963 **3.1.11 Batch Notification**

3964 Whereas batch servers are persistent entities, clients are often transient. For example, the *qsub*
 3965 utility creates a batch job and exits. For this reason, batch servers notify users of batch job events
 3966 by sending mail to the user that owns the job, or to other designated users.

3967 **3.2 Batch Services**

3968 The presence of Batch Environment Services and Utilities option services is indicated by the
 3969 configuration variable POSIX2_PBS. A conforming batch server provides services as defined in
 3970 this section. |

3971 A batch server shall provide batch services in two ways:

- 3972 1. The batch server provides a service at the request of a client.
- 3973 2. The batch server provides a deferred service as a result of a change in conditions
 3974 monitored by the batch server.

3975 If a batch server cannot complete a request, it shall reject the request. If a batch server cannot
 3976 complete a deferred service for a batch job, the batch server shall abort the batch job. Table 3-2
 3977 (on page 105) is a summary of environment variables that shall be supported by an
 3978 implementation of the batch server and utilities.

Table 3-2 Environment Variable Summary

Variable	Description
<i>PBS_DPREFIX</i>	Defines the directive prefix (see <i>qsub</i>)
<i>PBS_ENVIRONMENT</i>	Batch Job is batch or interactive (see Section 3.2.2.1)
<i>PBS_JOBID</i>	The <i>job_identifier</i> attribute of job (see Section 3.2.3.8)
<i>PBS_JOBNAME</i>	The <i>job_name</i> attribute of job (see Section 3.2.3.8)
<i>PBS_O_HOME</i>	Defines the <i>HOME</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_HOST</i>	Defines the host name of the batch client (see <i>qsub</i>)
<i>PBS_O_LANG</i>	Defines the <i>LANG</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_LOGNAME</i>	Defines the <i>LOGNAME</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_MAIL</i>	Defines the <i>MAIL</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_PATH</i>	Defines the <i>PATH</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_QUEUE</i>	Defines the submit queue of the batch client (see <i>qsub</i>)
<i>PBS_O_SHELL</i>	Defines the <i>SHELL</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_TZ</i>	Defines the <i>TZ</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_WORKDIR</i>	Defines the working directory of the batch client (see <i>qsub</i>)
<i>PBS_QUEUE</i>	Defines the initial execution queue (see Section 3.2.2.1)

3.2.1 Batch Job States

A batch job shall always be in one of the following states: QUEUED, RUNNING, HELD, WAITING, EXITING, or TRANSITING. The state of a batch job determines the types of requests that the batch server that manages the batch job can accept for the batch job. A batch server shall change the state of a batch job either in response to service requests from clients or as a result of deferred services, such as job execution or job routing.

A batch job that is in the QUEUED state resides in a queue but is still pending either execution or routing, depending on the queue type.

A batch server that queues a batch job in a routing queue shall put the batch job in the QUEUED state. A batch server that puts a batch job in an execution queue, but has not yet executed the batch job, shall put the batch job in the QUEUED state. A batch job that resides in an execution queue and is executing is defined to be in the RUNNING state. While a batch job is in the RUNNING state, a session leader is associated with the batch job.

A batch job that resides in an execution queue, but is ineligible to run because of a hold attribute, is defined to be in the HELD state.

A batch job that is not held, but must wait until a future date and time before executing, is defined to be in the WAITING state.

When the session leader associated with a running job exits, the batch job shall be placed in the EXITING state.

A batch job for which the session leader has terminated is defined to be in the EXITING state, and the batch server that manages such a batch job cannot accept job modification requests that affect the batch job. While a batch job is in the EXITING state, the batch server that manages the batch job is staging output files and notifying clients of job completion. Once a batch job has exited, it no longer exists as an object managed by a batch server.

A batch job that is being moved from a routing queue to another queue is defined to be in the TRANSITING state.

When a batch job in a routing queue has been selected to be moved to a new destination, then the batch job shall be in either the QUEUED state or the TRANSITING state, depending on the batch server implementation.

Batch jobs with either an *Execution_Time* attribute value set in the future or a *Hold_Types* attribute of value not equal to NO_HOLD, or both, may be routed or held in the routing queue. The treatment of jobs with the *Execution_Time* or *Hold_Types* attributes in a routing queue is implementation-defined.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has a *Hold_Types* attribute value of other than NO_HOLD, then the job should be in the HELD state.

Note: The effect of a hold upon a batch job in a routing queue is implementation-defined. The implementation should use the state that matches whether the batch job can route with a hold or not.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has:

- A *Hold_Types* attribute value of NO_HOLD
- An *Execution_Time* attribute in the past

then the batch job shall be in the QUEUED state.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has:

- A *Hold_Types* attribute value of NO_HOLD
- An *Execution_Time* attribute in the future

then the batch job may be in the WAITING state.

Note: The effect of a future execution time upon a batch job in a routing queue is implementation-defined. The implementation should use the state that matches whether the batch job can route with a hold or not.

Table 3-3 (on page 107) describes the next state of a batch job, given the current state of the batch job and the type of request. Table 3-4 (on page 108) describes the response of a batch server to a request, given the current state of the batch job and the type of request.

3.2.2 Deferred Batch Services

This section describes the deferred services performed by batch servers: job execution, job routing, job exit, job abort, and the rerunning of jobs after a restart.

3.2.2.1 Batch Job Execution

To execute a batch job is to create a session leader (a process) that runs the shell program indicated by the *Shell_Path_List* attribute of the batch job. The script is passed to the program as its standard input. An implementation may pass the script to the program by other implementation-defined means. At the time a batch job begins execution, it is defined to enter the RUNNING state.

Table 3-3 Next State Table

Request Type	Current State						
	X	Q	R	H	W	E	T
<i>Queue Batch Job Request</i>	Q	e	e	e	e	e	e
<i>Modify Batch Job Request</i>	e	Q	R	H	W	e	T
<i>Delete Batch Job Request</i>	e	X	E	X	X	E	X
<i>Batch Job Message Request</i>	e	Q	R	H	W	E	T
<i>Rerun Batch Job Request</i>	e	e	Q	e	e	e	e
<i>Signal Batch Job Request</i>	e	e	R	H	W	e	e
<i>Batch Job Status Request</i>	e	Q	R	H	W	E	T
<i>Batch Queue Status Request</i>	X	Q	R	H	W	E	T
<i>Server Status Request</i>	X	Q	R	H	W	E	T
<i>Select Batch Jobs Request</i>	X	Q	R	H	W	E	T
<i>Move Batch Job Request</i>	e	Q	R	H	W	e	T
<i>Hold Batch Job Request</i>	e	H	R / H	H	H	e	T
<i>Release Batch Job Request</i>	e	Q	R	Q / W / H	W	e	T
<i>Server Shutdown Request</i>	X	Q	Q	H	W	E	T
<i>Locate Batch Job Request</i>	e	Q	R	H	W	E	T

Legend

X Nonexistent

Q QUEUED

R RUNNING

H HELD

W WAITING

E EXITING

T TRANSITING

e Error

A batch server that has an execution queue containing jobs is said to own the queue and manage the batch jobs in that queue. A batch server that has been started shall execute the batch jobs in the execution queues owned by the batch server. The batch server shall schedule for execution those jobs in the execution queues that are in the QUEUED state. The algorithm for scheduling jobs is implementation-defined.

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, an environment variable named *PBS_ENVIRONMENT*, the value of which is the string *PBS_BATCH* encoded in the portable character set.

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, an environment variable named *PBS_QUEUE*, the value of which is the name of the execution queue of the batch job encoded in the portable character set.

To rerun a batch job is to requeue a batch job that is currently executing and then kill the session leader of the executing job by sending a SIGKILL prior to completion; see Section 3.2.3.11 (on page 120). A batch server that reruns a batch job shall append the standard output and standard error files of the batch job to the corresponding files of the previous execution, if they exist, with appropriate annotation. If either file does not exist, that file shall be created as in normal execution.

Table 3-4 Results / Output Table

Request Type	Current State						
	X	Q	R	H	W	E	T
Queue Batch Job Request	O	e	e	e	e	e	e
Modify Batch Job Request	e	O	e	O	O	e	e
Delete Batch Job Request	e	O	O	O	O	e	O
Batch Job Message Request	e	e	O	e	e	e	e
Rerun Batch Job Request	e	e	O	e	e	e	e
Signal Batch Job Request	e	e	O	e	e	e	e
Batch Job Status Request	e	O	O	O	O	O	O
Batch Queue Status Request	O	O	O	O	O	O	O
Server Status Request	O	O	O	O	O	O	O
Select Batch Job Request	e	O	O	O	O	O	O
Move Batch Job Request	e	O	O	O	O	e	e
Hold Batch Job Request	e	O	O	O	O	e	e
Release Batch Job Request	e	O	e	O	O	e	e
Server Shutdown Request	O	O	e	O	O	e	e
Locate Batch Job Request	e	O	O	O	O	O	O

Legend

O OK

e Error message

The execution of a batch job by a batch server shall be controlled by job, queue, and server attributes, as defined in this section.

Account_Name Attribute

Batch accounting is an optional feature of batch servers. If a batch server implements accounting, the statements in this section apply and the configuration variable `POSIX2_PBS_ACCOUNTING` shall be set to 1.

A batch server that executes a batch job shall charge the account named in the *Account_Name* attribute of the batch job for resources consumed by the batch job.

If the *Account_Name* attribute of the batch job is absent from the batch job attribute list or is altered while the batch job is in execution, the batch server action is implementation-defined.

Checkpoint Attribute

Batch checkpointing is an optional feature of batch servers. If a batch server implements checkpointing, the statements in this section apply and the configuration variable `POSIX2_PBS_CHECKPOINT` shall be set to 1.

There are two attributes associated with the checkpointing feature: *Checkpoint* and *Minimum_Cpu_Interval*. *Checkpoint* is a batch job attribute, while *Minimum_Cpu_Interval* is a queue attribute. An implementation that does not support checkpointing shall support the *Checkpoint* job attribute to the extent that the batch server shall maintain and pass this attribute to other servers.

The behavior of a batch server that executes a batch job for which the value of the *Checkpoint* attribute is `CHECKPOINT_UNSPECIFIED` is implementation-defined. A batch server that executes a batch job for which the value of the *Checkpoint* attribute is `NO_CHECKPOINT` shall

4147 not checkpoint the batch job.

4148 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is
4149 CHECKPOINT_AT_SHUTDOWN shall checkpoint the batch job only when the batch server
4150 accepts a request to shut down during the time when the batch job is in the RUNNING state.

4151 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is
4152 CHECKPOINT_AT_MIN_CPU_INTERVAL shall checkpoint the batch job at the interval
4153 specified by the *Minimum_Cpu_Interval* attribute of the queue for which the batch job has been
4154 selected. The *Minimum_Cpu_Interval* attribute shall be specified in units of CPU minutes.

4155 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is an
4156 unsigned integer shall checkpoint the batch job at an interval that is the value of either the
4157 *Checkpoint* attribute, or the *Minimum_Cpu_Interval* attribute of the queue for which the batch job
4158 has been selected, whichever is greater. Both intervals shall be in units of CPU minutes. When
4159 the *Minimum_Cpu_Interval* attribute is greater than the *Checkpoint* attribute, the batch job shall
4160 write a warning message to the standard error stream of the batch job.

4161 **Error_Path Attribute**

4162 The *Error_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When
4163 the *Join_Path* attribute of the batch job is set to the value FALSE and the *Keep_Files* attribute of
4164 the batch job does not contain the value KEEP_STD_ERROR, a batch server that executes a batch
4165 job shall perform one of the following actions:

- 4166 • Set the standard error stream of the session leader of the batch job to the path described by
4167 the value of the *Error_Path* attribute of the batch job.
- 4168 • Buffer the standard error of the session leader of the batch job until completion of the batch
4169 job, and when the batch job exits return the contents to the destination described by the value
4170 of the *Error_Path* attribute of the batch job.

4171 Applications shall not rely on having access to the standard error of a batch job prior to the
4172 completion of the batch job.

4173 When the *Error_Path* attribute does not specify a host name, then the batch server shall retain the
4174 standard error of the batch job on the host of execution.

4175 When the *Error_Path* attribute does specify a host name and the *Keep_Files* attribute does not
4176 contain the value KEEP_STD_ERROR, then the final destination of the standard error of the
4177 batch job shall be on the host whose host name is specified.

4178 If the path indicated by the value of the *Error_Path* attribute of the batch job is a relative path, the
4179 batch server shall expand the path relative to the home directory of the user on the host to which
4180 the file is being returned.

4181 When the batch server buffers the standard error of the batch job and the file cannot be opened
4182 for write upon completion of the batch job, then the server shall place the standard error in an
4183 implementation-defined location and notify the user of the location via mail. It shall be possible
4184 for the user to process this mail using the *mailx* utility.

4185 If a batch server that does not buffer the standard error cannot open the standard error path of
4186 the batch job for write access, then the batch server shall abort the batch job.

Execution_Time Attribute

A batch server shall not execute a batch job before the time represented by the value of the *Execution_Time* attribute of the batch job. The *Execution_Time* attribute is defined in seconds since the Epoch.

Hold_Types Attribute

A batch server shall support the following hold types:

- s Can be set or released by a user with at least a privilege level of batch administrator (SYSTEM).
- o Can be set or released by a user with at least a privilege level of batch operator (OPERATOR).
- u Can be set or released by the user with at least a privilege level of user, where the user is defined in the *Job_Owner* attribute (USER).
- n Indicates that none of the *Hold_Types* attributes are set (NO_HOLD).

An implementation may define other hold types. Any additional hold types, how they are specified, their internal representation, their behavior, and how they affect the behavior of other utilities are implementation-defined.

The value of the *Hold_Types* attribute shall be the union of the valid hold types ('s', 'o', 'u', and any implementation-defined hold types), or 'n'.

A batch server shall not execute a batch job if the *Hold_Types* attribute of the batch job has a value other than NO_HOLD. If the *Hold_Types* attribute of the batch job has a value other than NO_HOLD, the batch job shall be in the HELD state.

Job_Owner Attribute

The *Job_Owner* attribute consists of a pair of user name and host name values of the form:

username@hostname

A batch server that accepts a *Queue Batch Job Request* shall set the *Job_Owner* attribute to a string that is the *username@hostname* of the user who submitted the job.

Join_Path Attribute

A batch server that executes a batch job for which the value of the *Join_Path* attribute is TRUE shall ignore the value of the *Error_Path* attribute and merge the standard error of the batch job with the standard output of the batch job.

Keep_Files Attribute

A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes the value KEEP_STD_OUTPUT shall retain the standard output of the batch job on the host where execution occurs. The standard output shall be retained in the home directory of the user under whose user ID the batch job is executed and the filename shall be the default filename for the standard output as defined under the **-o** option of the *qsub* utility. The *Output_Path* attribute is not modified.

A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes the value KEEP_STD_ERROR shall retain the standard error of the batch job on the host where execution occurs. The standard error shall be retained in the home directory of the user under whose user ID the batch job is executed and the filename shall be the default filename for

4228 standard error as defined under the `-e` option of the `qsub` utility. The *Error_Path* attribute is not
4229 modified.

4230 A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes
4231 values other than `KEEP_STD_OUTPUT` and `KEEP_STD_ERROR` shall retain these other files on
4232 the host where execution occurs. These files (with implementation-defined names) shall be
4233 retained in the home directory of the user under whose user identifier the batch job is executed.

4234 **Mail_Points and Mail_Users Attributes**

4235 A batch server that executes a batch job for which one of the values of the *Mail_Points* attribute is
4236 the value `MAIL_AT_BEGINNING` shall send a mail message to each user account listed in the
4237 *Mail_Users* attribute of the batch job.

4238 The mail message shall contain at least the batch job identifier, queue, and server at which the
4239 batch job currently resides, and the *Job_Owner* attribute.

4240 **Output_Path Attribute**

4241 The *Output_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*.
4242 When the *Keep_Files* attribute of the batch job does not contain the value `KEEP_STD_OUTPUT`, a
4243 batch server that executes a batch job shall either:

4244 • Set the standard output stream of the session leader of the batch job to the destination
4245 described by the value of the *Output_Path* attribute of the batch job.

4246 or:

4247 • Buffer the standard output of the session leader of the batch job until completion of the batch
4248 job, and when the batch job exits return the contents to the destination described by the value
4249 of the *Output_Path* attribute of the batch job.

4250 When the *Output_Path* attribute does not specify a host name, then the batch server shall retain
4251 the standard output of the batch job on the host of execution.

4252 When the *Keep_Files* attribute does not contain the value `KEEP_STD_OUTPUT` and the
4253 *Output_Path* attribute does specify a host name, then the final destination of the standard output
4254 of the batch job shall be on the host specified.

4255 If the path specified in the *Output_Path* attribute of the batch job is a relative path, the batch
4256 server shall expand the path relative to the home directory of the user on the host to which the
4257 file is being returned.

4258 Whether or not the batch server buffers the standard output of the batch job until completion of
4259 the batch job is implementation-defined. Applications shall not rely on having access to the
4260 standard output of a batch job prior to the completion of the batch job.

4261 When the batch server does buffer the standard output of the batch job and the file cannot be
4262 opened for write upon completion of the batch job, then the batch server shall place the standard
4263 output in an implementation-defined location and notify the user of the location via mail. It shall
4264 be possible for the user to process this mail using the *mailx* utility.

4265 If a batch server that does not buffer the standard output cannot open the standard output path
4266 of the batch job for write access, then the batch server shall abort the batch job.

Priority Attribute

A batch server implementation may choose to preferentially execute a batch job based on the *Priority* attribute. The interpretation of the batch job *Priority* attribute by a batch server is implementation-defined. If an implementation uses the *Priority* attribute, it shall interpret larger values of the *Priority* attribute to mean the batch job shall be preferentially selected for execution.

Rerunable Attribute

A batch job that began execution but did not complete, because the batch server either shut down or terminated abnormally, shall be requeued if the *Rerunable* attribute of the batch job has the value TRUE.

If a batch job, which was requeued after beginning execution but prior to completion, has a valid checkpoint file and the batch server supports checkpointing, then the batch job shall be restarted from the last valid checkpoint.

If the batch job cannot be restarted from a checkpoint, then when a batch job has a *Rerunable* attribute value of TRUE and was requeued after beginning execution but prior to completion, the batch server shall place the batch job into execution at the beginning of the job.

When a batch job has a *Rerunable* attribute value other than TRUE and was requeued after beginning execution but prior to completion, and the batch job cannot be restarted from a checkpoint, then the batch server shall abort the batch job.

Resource_List Attribute

A batch server that executes a batch job shall establish the resource limits of the session leader of the batch job according to the values of the *Resource_List* attribute of the batch job. Resource limits shall be enforced by an implementation-defined method.

Shell_Path_List Attribute

The *Shell_Path_List* job attribute consists of a list of pairs of pathname and host name values. The host name component can be omitted, in which case the pathname serves as the default pathname when a batch server cannot find the name of the host on which it is running in the list.

A batch server that executes a batch job shall select, from the value of the *Shell_Path_List* attribute of the batch job, a pathname where the shell to execute the batch job shall be found. The batch server shall select the pathname, in order of preference, according to the following methods:

- Select the pathname that contains the name of the host on which the batch server is running.
- Select the pathname for which the host name has been omitted.
- Select the pathname for the login shell of the user under which the batch job is to execute.

If the shell path value selected is an invalid pathname, the batch server shall abort the batch job.

If the value of the selected pathname from the *Shell_Path_List* attribute of the batch job represents a partial path, the batch server shall expand the path relative to a path that is implementation-defined.

The batch server that executes the batch job shall execute the program that was selected from the *Shell_Path_List* attribute of the batch job. The batch server shall pass the path to the script of the batch job as the first argument to the shell program.

User_List Attribute

The *User_List* job attribute consists of a list of pairs of user name and host name values. The host name component can be omitted, in which case the user name serves as a default when a batch server cannot find the name of the host on which it is running in the list.

A batch server that executes a batch job shall select, from the value of the *User_List* attribute of the batch job, a user name under which to create the session leader. The server shall select the user name, in order of preference, according to the following methods:

- Select the user name of a value that contains the name of the host on which the batch server executes.
- Select the user name of a value for which the host name has been omitted.
- Select the user name from the *Job_Owner* attribute of the batch job.

Variable_List Attribute

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, each environment variable listed in the *Variable_List* attribute of the batch job, and set the value of each such environment variable to that of the corresponding variable in the variable list.

3.2.2.2 Batch Job Routing

To route a batch job is to select a queue from a list and move the batch job to that queue.

A batch server that has routing queues, which have been started, shall route the jobs in the routing queues owned by the batch server. A batch server may delay the routing of a batch job. The algorithm for selecting a batch job and the queue to which it will be routed is implementation-defined.

When a routing queue has multiple possible destinations specified, then the precedence of the destinations is implementation-defined.

A batch server that routes a batch job to a queue at another server shall move the batch job into the target queue with a *Queue Batch Job Request*.

If the target server rejects the *Queue Batch Job Request*, the routing server shall retry routing the batch job or abort the batch job. A batch server that retries failed routings shall provide a means for the batch administrator to specify the number of retries and the minimum period of time between retries. The means by which an administrator specifies the number of retries and the delay between retries is implementation-defined. When the number of retries specified by the batch administrator has been exhausted, the batch server shall abort the batch job and perform the functions of *Batch Job Exit*; see Section 3.2.2.3.

3.2.2.3 Batch Job Exit

For each job in the EXITING state, the batch server that exited the batch job shall perform the following deferred services in the order specified:

1. If buffering standard error, move that file into the location specified by the *Error_Path* attribute of the batch job.
2. If buffering standard output, move that file into the location specified by the *Output_Path* attribute of the batch job.
3. If the *Mail_Points* attribute of the batch job includes MAIL_AT_EXIT, send mail to the users listed in the *Mail_Users* attribute of the batch job. The mail message shall contain at least

4349 the batch job identifier, queue, and server at which the batch job currently resides, and the
 4350 *Job_Owner* attribute.

4351 4. Remove the batch job from the queue.

4352 If a batch server that buffers the standard error output cannot return the standard error file to
 4353 the standard error path at the time the batch job exits, the batch server shall do one of the
 4354 following:

- 4355 • Mail the standard error file to the batch job owner.
- 4356 • Save the standard error file and mail the location and name of the file where the standard
 4357 error is stored to the batch job owner.
- 4358 • Save the standard error file and notify the user by other implementation-defined means.

4359 If a batch server that buffers the standard output cannot return the standard output file to the
 4360 standard output path at the time the batch job exits, the batch server shall do one of the
 4361 following:

- 4362 • Mail the standard output file to the batch job owner.
- 4363 • Save the standard output file and mail the location and name of the file where the standard
 4364 output is stored to the batch job owner.
- 4365 • Save the standard output file and notify the user by other implementation-defined means.

4366 At the conclusion of job exit processing, the batch job is no longer managed by a batch server.

4367 3.2.2.4 *Batch Server Restart*

4368 A batch server that has been either shutdown or terminated abnormally, and has returned to
 4369 operation, is said to have “restarted”.

4370 Upon restarting, a batch server shall requeue those jobs managed by the batch server that were
 4371 in the RUNNING state at the time the batch server shut down and for which the *Rerunable*
 4372 attribute of the batch job has the value TRUE.

4373 Queues are defined to be non-volatile. A batch server shall store the content of queues that it
 4374 controls in such a way that server and system shutdowns do not erase the content of the queues.

4375 3.2.2.5 *Batch Job Abort*

4376 A batch server that cannot perform a deferred service for a batch job shall abort the batch job.

4377 A batch server that aborts a batch job shall perform the following services:

- 4378 • Delete the batch job from the queue in which it resides.
- 4379 • If the *Mail_Points* attribute of the batch job includes the value MAIL_AT_ABORT, send mail
 4380 to the users listed in the value of the *Mail_Users* attribute of the job. The mail message shall
 4381 contain at least the batch job identifier, queue, and server at which the batch job currently
 4382 resides, the *Job_Owner* attribute, and the reason for the abort.
- 4383 • If the batch job was in the RUNNING state, terminate the session leader of the executing job
 4384 by sending the session leader a SIGKILL, place the batch job in the EXITING state, and
 4385 perform the actions of *Batch Job Exit*.

3.2.3 Requested Batch Services

This section describes the services provided by batch servers in response to requests from clients. Table 3-5 summarizes the current set of batch service requests and for each gives its type (deferred or not) and whether it is an optional function.

Table 3-5 Batch Services Summary

Batch Service	Deferred	Optional
<i>Batch Job Execution</i>	Yes	No
<i>Batch Job Routing</i>	Yes	No
<i>Batch Job Exit</i>	Yes	No
<i>Batch Server Restart</i>	Yes	No
<i>Batch Job Abort</i>	Yes	No
<i>Delete Batch Job Request</i>	No	No
<i>Hold Batch Job Request</i>	No	No
<i>Batch Job Message Request</i>	No	Yes
<i>Batch Job Status Request</i>	No	No
<i>Locate Batch Job Request</i>	No	Yes
<i>Modify Batch Job Request</i>	No	No
<i>Move Batch Job Request</i>	No	No
<i>Queue Batch Job Request</i>	No	No
<i>Batch Queue Status Request</i>	No	No
<i>Release Batch Job Request</i>	No	No
<i>Rerun Batch Job Request</i>	No	No
<i>Select Batch Jobs Request</i>	No	No
<i>Server Shutdown Request</i>	No	No
<i>Server Status Request</i>	No	No
<i>Signal Batch Job Request</i>	No	No
<i>Track Batch Job Request</i>	No	Yes

If a request is rejected because the batch client is not authorized to perform the action, the batch server shall return the same status as when the batch job does not exist.

3.2.3.1 Delete Batch Job Request

A batch job is defined to have been deleted when it has been removed from the queue in which it resides and not instantiated in another queue. A client requests that the server that manages a batch job delete the batch job. Such a request is called a *Delete Batch Job Request*.

A batch server shall reject a *Delete Batch Job Request* if any of the following statements are true:

- The user of the batch client is not authorized to delete the designated job.
- The designated job is not managed by the batch server.
- The designated job is in a state inconsistent with the delete request.

A batch server may reject a *Delete Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server requested to delete a batch job shall delete the batch job if the batch job exists and is not in the EXITING state.

A batch server that deletes a batch job in the RUNNING state shall send a SIGKILL signal to the session leader of the batch job. It is implementation-defined whether additional signals are sent

4430 to the session leader of the job prior to sending the SIGKILL signal.

4431 A batch server that deletes a batch job in the RUNNING state shall place the batch job in the

4432 EXITING state after it has killed the session leader of the batch job and shall perform the actions

4433 of *Batch Job Exit*.

4434 3.2.3.2 *Hold Batch Job Request*

4435 A batch client can request that the batch server add one or more holds to a batch job. Such a

4436 request is called a *Hold Batch Job Request*.

4437 A batch server shall reject a *Hold Batch Job Request* if any of the following statements are true:

- 4438 • The batch server does not support one or more of the requested holds to be added to the
- 4439 batch job.
- 4440 • The user of the batch client is not authorized to add one or more of the requested holds to the
- 4441 batch job.
- 4442 • The batch server does not manage the specified job.
- 4443 • The designated job is in the EXITING state.

4444 A batch server may reject a *Hold Batch Job Request* for other implementation-defined reasons. The

4445 method used to determine whether the user of a client is authorized to perform the requested

4446 action is implementation-defined.

4447 A batch server that accepts a *Hold Batch Job Request* for a batch job in the RUNNING state shall

4448 place a hold on the batch job. The effects, if any, the hold will have on a batch job in the

4449 RUNNING state are implementation-defined.

4450 A batch server that accepts a *Hold Batch Job Request* shall add each type of hold listed in the *Hold*

4451 *Batch Job Request*, that is not already present, to the value of the *Hold_Types* attribute of the batch

4452 job.

4453 3.2.3.3 *Batch Job Message Request*

4454 *Batch Job Message Request* is an optional feature of batch servers. If an implementation supports

4455 *Batch Job Message Request*, the statements in this section apply and the configuration variable

4456 POSIX2_PBS_MESSAGE shall be set to 1.

4457 A batch client can request that a batch server write a message into certain output files of a batch

4458 job. Such a request is called a *Batch Job Message Request*.

4459 A batch server shall reject a *Batch Job Message Request* if any of the following statements are true:

- 4460 • The batch server does not support sending messages to jobs.
- 4461 • The user of the batch client is not authorized to post a message to the designated job.
- 4462 • The designated job does not exist on the batch server.
- 4463 • The designated job is not in the RUNNING state.

4464 A batch server may reject a *Batch Job Message Request* for other implementation-defined reasons.

4465 The method used to determine whether the user of a client is authorized to perform the

4466 requested action is implementation-defined.

4467 A batch server that accepts a *Batch Job Message Request* shall write the message sent by the batch

4468 client into the files indicated by the batch client.

4469 3.2.3.4 *Batch Job Status Request*

4470 A batch client can request that a batch server respond with the status and attributes of a batch
4471 job. Such a request is called a *Batch Job Status Request*.

4472 A batch server shall reject a *Batch Job Status Request* if any of the following statements are true:

- 4473 • The user of the batch client is not authorized to query the status of the designated job.
- 4474 • The designated job is not managed by the batch server.

4475 A batch server may reject a *Batch Job Status Request* for other implementation-defined reasons.
4476 The method used to determine whether the user of a client is authorized to perform the
4477 requested action is implementation-defined.

4478 A batch server that accepts a *Batch Job Status Request* shall return a *Batch Job Status Message* to the
4479 batch client.

4480 A batch server may return other information in response to a *Batch Job Status Request*.

4481 3.2.3.5 *Locate Batch Job Request*

4482 *Locate Batch Job Request* is an optional feature of batch servers. If an implementation supports
4483 *Locate Batch Job Request*, the statements in this section apply and the configuration variable
4484 POSIX2_PBS_LOCATE shall be set to 1.

4485 A batch client can ask a batch server to respond with the location of a batch job that was created
4486 by the batch server. Such a request is called a *Locate Batch Job Request*.

4487 A batch server that accepts a *Locate Batch Job Request* shall return a *Batch Job Location Message* to
4488 the batch client.

4489 A batch server may reject a *Locate Batch Job Request* for a batch job that was not created by that
4490 server.

4491 A batch server may reject a *Locate Batch Job Request* for a batch job that is no longer managed by
4492 that server; that is, for a batch job that is not in a queue owned by that server.

4493 A batch server may reject a *Locate Batch Job Request* for other implementation-defined reasons.

4494 3.2.3.6 *Modify Batch Job Request*

4495 Batch clients modify (alter) the attributes of a batch job by making a request to the server that
4496 manages the batch job. Such a request is called a *Modify Batch Job Request*.

4497 A batch server shall reject a *Modify Batch Job Request* if any of the following statements are true:

- 4498 • The user of the batch client is not authorized to make the requested modification to the batch
4499 job.
- 4500 • The designated job is not managed by the batch server.
- 4501 • The requested modification is inconsistent with the state of the batch job.
- 4502 • An unrecognized resource is requested for a batch job in an execution queue.

4503 A batch server may reject a *Modify Batch Job Request* for other implementation-defined reasons.
4504 The method used to determine whether the user of a client is authorized to perform the
4505 requested action is implementation-defined.

4506 A batch server that accepts a *Modify Batch Job Request* shall modify all the specified attributes of
4507 the batch job. A batch server that rejects a *Modify Batch Job Request* shall modify none of the
4508 attributes of the batch job.

4509 If the servicing by a batch server of an otherwise valid request would result in no change, then
 4510 the batch server shall indicate successful completion of the request.

4511 3.2.3.7 *Move Batch Job Request*

4512 A batch client can request that a batch server move a batch job to another destination. Such a
 4513 request is called a *Move Batch Job Request*.

4514 A batch server shall reject a *Move Batch Job Request* if any of the following statements are true:

- 4515 • The user of the batch client is not authorized to remove the designated job from the queue in
 4516 which the batch job resides.
- 4517 • The user of the batch client is not authorized to move the designated job to the destination.
- 4518 • The designated job is not managed by the batch server.
- 4519 • The designated job is in the EXITING state.
- 4520 • The destination is inaccessible.

4521 A batch server can reject a *Move Batch Job Request* for other implementation-defined reasons. The
 4522 method used to determine whether the user of a client is authorized to perform the requested
 4523 action is implementation-defined.

4524 A batch server that accepts a *Move Batch Job Request* shall perform the following services:

- 4525 • Queue the designated job at the destination.
- 4526 • Remove the designated job from the queue in which the batch job resides.

4527 If the destination resides on another batch server, the batch server shall queue the batch job at
 4528 the destination by sending a *Queue Batch Job Request* to the other server. If the *Queue Batch Job*
 4529 *Request* fails, the batch server shall reject the *Move Batch Job Request*. If the *Queue Batch Job Request*
 4530 succeeds, the batch server shall remove the batch job from its queue.

4531 The batch server shall not modify any attributes of the batch job.

4532 3.2.3.8 *Queue Batch Job Request*

4533 A batch queue is controlled by one and only one batch server. A batch server is said to own the
 4534 queues that it controls. Batch clients make requests of batch servers to have jobs queued. Such a
 4535 request is called a *Queue Batch Job Request*.

4536 A batch server requested to queue a batch job for which the queue is not specified shall select an
 4537 implementation-defined queue for the batch job. Such a queue is called the “default queue” of
 4538 the batch server. The implementation shall provide the means for a batch administrator to
 4539 specify the default queue. The queue, whether specified or defaulted, is called the “target
 4540 queue”.

4541 A batch server shall reject a *Queue Batch Job Request* if any of the following statements are true:

- 4542 • The client is not authorized to create a batch job in the target queue.
- 4543 • The request specifies a queue that does not exist on the batch server.
- 4544 • The target queue is an execution queue and the batch server cannot satisfy a resource
 4545 requirement of the batch job.
- 4546 • The target queue is an execution queue and an unrecognized resource is requested.
- 4547 • The target queue is an execution queue, the batch server does not support checkpointing, and
 4548 the value of the *Checkpoint* attribute of the batch job is not NO_CHECKPOINT.

- 4549 • The job requires access to a user identifier that the batch client is not authorized to access.
- 4550 A batch server may reject a *Queue Batch Job Request* for other implementation-defined reasons.
- 4551 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the
- 4552 PBS_O_QUEUE value is missing from the value of the *Variable_List* attribute of the batch job
- 4553 shall add that variable to the list and set the value to the name of the target queue. Once set, no
- 4554 server shall change the value of PBS_O_QUEUE, even if the batch job is moved to another
- 4555 queue.
- 4556 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS_JOBID
- 4557 value is missing from the value of the *Variable_List* attribute shall add that variable to the list and
- 4558 set the value to the batch job identifier assigned by the server in the format:
- 4559 sequence_number.server
- 4560 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the
- 4561 PBS_JOBNAME value is missing from the value of the *Variable_List* attribute of the batch job
- 4562 shall add that variable to the list and set the value to the *Job_Name* attribute of the batch job.
- 4563 3.2.3.9 *Batch Queue Status Request*
- 4564 A batch client can request that a batch server respond with the status and attributes of a queue.
- 4565 Such a request is called a *Batch Queue Status Request*.
- 4566 A batch server shall reject a *Batch Queue Status Request* if any of the following statements are true:
- 4567 • The user of the batch client is not authorized to query the status of the designated queue.
- 4568 • The designated queue does not exist on the batch server.
- 4569 A batch server may reject a *Batch Queue Status Request* for other implementation-defined reasons.
- 4570 The method used to determine whether the user of a client is authorized to perform the
- 4571 requested action is implementation-defined.
- 4572 A batch server that accepts a *Batch Queue Status Request* shall return a *Batch Queue Status Reply* to
- 4573 the batch client.
- 4574 3.2.3.10 *Release Batch Job Request*
- 4575 A batch client can request that the server remove one or more holds from a batch job. Such a
- 4576 request is called a *Release Batch Job Request*.
- 4577 A batch server shall reject a *Release Batch Job Request* if any of the following statements are true:
- 4578 • The user of the batch client is not authorized to remove one or more of the requested holds
- 4579 from the batch job.
- 4580 • The batch server does not manage the specified job.
- 4581 A batch server may reject a *Release Batch Job Request* for other implementation-defined reasons.
- 4582 The method used to determine whether the user of a client is authorized to perform the
- 4583 requested action is implementation-defined.
- 4584 A batch server that accepts a *Release Batch Job Request* shall remove each type of hold listed in the
- 4585 *Release Batch Job Request*, that is present, from the value of the *Hold_Types* attribute of the batch
- 4586 job.

4587 3.2.3.11 *Rerun Batch Job Request*

4588 To rerun a batch job is to kill the session leader of the batch job and leave the batch job eligible
4589 for re-execution. A batch client can request that a batch server rerun a batch job. Such a request is
4590 called *Rerun Batch Job Request*.

4591 A batch server shall reject a *Rerun Batch Job Request* if any of the following statements are true:

- 4592 • The user of the batch client is not authorized to rerun the designated job.
- 4593 • The *Rerunable* attribute of the designated job has the value FALSE.
- 4594 • The designated job is not in the RUNNING state.
- 4595 • The batch server does not manage the designated job.

4596 A batch server may reject a *Rerun Batch Job Request* for other implementation-defined reasons.
4597 The method used to determine whether the user of a client is authorized to perform the
4598 requested action is implementation-defined.

4599 A batch server that rejects a *Rerun Batch Job Request* shall in no way modify the execution of the
4600 batch job.

4601 A batch server that accepts a request to rerun a batch job shall perform the following services:

- 4602 • Requeue the batch job in the execution queue in which it was executing.
- 4603 • Send a SIGKILL signal to the process group of the session leader of the batch job.

4604 An implementation may indicate to the batch job owner that the batch job has been rerun.
4605 Whether and how the batch job owner is notified that a batch job is rerun is implementation-
4606 defined.

4607 A batch server that reruns a batch job may send other implementation-defined signals to the
4608 session leader of the batch job prior to sending the SIGKILL signal.

4609 A batch server may preferentially select a rerun job for execution. Whether rerun jobs shall be
4610 selected for execution before other jobs is implementation-defined.

4611 3.2.3.12 *Select Batch Jobs Request*

4612 A batch client can request from a batch server a list of jobs managed by that server that match a
4613 list of selection criteria. Such a request is called a *Select Batch Jobs Request*. All the batch jobs
4614 managed by the batch server that receives the request are candidates for selection.

4615 A batch server that accepts a *Select Batch Jobs Request* shall return a list of zero or more job
4616 identifiers that correspond to jobs that meet the selection criteria.

4617 If the batch client is not authorized to query the status of a batch job, the batch server shall not
4618 select the batch job.

4619 3.2.3.13 *Server Shutdown Request*

4620 A batch server is defined to have shut down when it does not respond to requests from clients
4621 and does not perform deferred services for jobs. A batch client can request that a batch server
4622 shut down. Such a request is called a *Server Shutdown Request*.

4623 A batch server shall reject a *Server Shutdown Request* from a client that is not authorized to shut
4624 down the batch server. The method used to determine whether the user of a client is authorized
4625 to perform the requested action is implementation-defined.

4626 A batch server may reject a *Server Shutdown Request* for other implementation-defined reasons.
 4627 The reasons for which a *Server Shutdown Request* may be rejected are implementation-defined.

4628 At server shutdown, a batch server shall do, in order of preference, one of the following:

- 4629 • If checkpointing is implemented and the batch job is checkpointable, then checkpoint the
 4630 batch job and requeue it.
- 4631 • If the batch job is rerunnable, then requeue the batch job to be rerun (restarted from the
 4632 beginning).
- 4633 • Abort the batch job.

4634 3.2.3.14 *Server Status Request*

4635 A batch client can request that a batch server respond with the status and attributes of the batch
 4636 server. Such a request is called a *Server Status Request*.

4637 A batch server shall reject a *Server Status Request* if the following statement is true:

- 4638 • The user of the batch client is not authorized to query the status of the designated server.

4639 A batch server may reject a *Server Status Request* for other implementation-defined reasons. The
 4640 method used to determine whether the user of a client is authorized to perform the requested
 4641 action is implementation-defined.

4642 A batch server that accepts a *Server Status Request* shall return a *Server Status Reply* to the batch
 4643 client.

4644 3.2.3.15 *Signal Batch Job Request*

4645 A batch client can request that a batch server signal the session leader of a batch job. Such a
 4646 request is called a *Signal Batch Job Request*.

4647 A batch server shall reject a *Signal Batch Job Request* if any of the following statements are true:

- 4648 • The user of the batch client is not authorized to signal the batch job.
- 4649 • The job is not in the RUNNING state.
- 4650 • The batch server does not manage the designated job.
- 4651 • The requested signal is not supported by the implementation.

4652 A batch server may reject a *Signal Batch Job Request* for other implementation-defined reasons.
 4653 The method used to determine whether the user of a client is authorized to perform the
 4654 requested action is implementation-defined.

4655 A batch server that accepts a request to signal a batch job shall send the signal requested by the
 4656 batch client to the process group of the session leader of the batch job.

4657 3.2.3.16 *Track Batch Job Request*

4658 *Track Batch Job Request* is an optional feature of batch servers. If an implementation supports
 4659 *Track Batch Job Request*, the statements in this section apply and the configuration variable
 4660 POSIX2_PBS_TRACK shall be set to 1.

4661 *Track Batch Job Request* provides a method for tracking the current location of a batch job. Clients
 4662 may use the tracking information to determine the batch server that should receive a batch
 4663 server request.

4664 If *Track Batch Job Request* is supported by a batch server, then when the batch server queues a
 4665 batch job as a result of a *Queue Batch Job Request*, and the batch server is not the batch server that
 4666 created the batch job, the batch server shall send a *Track Batch Job Request* to the batch server that
 4667 created the job.

4668 If *Track Batch Job Request* is supported by a batch server, then the *Track Batch Job Request* may also
 4669 be sent to other servers as a backup to the primary server. The method by which backup servers
 4670 are specified is implementation-defined.

4671 If *Track Batch Job Request* is supported by a batch server that receives a *Track Batch Job Request*,
 4672 then the batch server shall record the current location of the batch job as contained in the
 4673 request.

4674 **3.3 Common Behavior for Batch Environment Utilities**

4675 **3.3.1 Batch Job Identifier**

4676 A utility shall recognize *job_identifiers* of the format:

4677 [sequence_number] [.server_name] [@server]

4678 where:

4679 *sequence_number* An integer that, when combined with *server_name*, provides a batch job
 4680 identifier that is unique within the batch system.

4681 *server_name* The name of the batch server to which the batch job was originally submitted.

4682 *server* The name of the batch server that is currently managing the batch job.

4683 If the application omits the batch *server_name* portion of a batch job identifier, a utility shall use
 4684 the name of a default batch server.

4685 If the application omits the batch *server* portion of a batch job identifier, a utility shall use:

- 4686 • The batch server indicated by *server_name*, if present
- 4687 • The name of the default batch server
- 4688 • The name of the batch server that is currently managing the batch job

4689 If only *@server* is specified, then the status of all jobs owned by the user on the requested server
 4690 is listed.

4691 The means by which a utility determines the default batch server is implementation-defined.

4692 If the application presents the batch *server* portion of a batch job identifier to a utility, the utility
 4693 shall send the request to the specified server.

4694 A strictly conforming application shall use the syntax described for the job identifier. Whenever
 4695 a batch job identifier is specified whose syntax is not recognized by an implementation, then a
 4696 message for each error that occurs shall be written to standard error and the utility shall exit
 4697 with an exit status greater than zero.

4698 When a batch job identifier is supplied as an argument to a batch utility and the *server_name*
 4699 portion of the batch job identifier is omitted, then the utility shall use the name of the default
 4700 batch server.

4701 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*
 4702 portion of the batch job identifier is omitted, then the utility shall use either:

- 4703 • The name of the default batch server
 - 4704 or:
 - 4705 • The name of the batch server that is currently managing the batch job
- 4706 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*
 4707 portion of the batch job identifier is specified, then the utility shall send the required *Batch Server*
 4708 *Request* to the specified server.

4709 3.3.2 Destination

4710 The utility shall recognize a *destination* of the format:

4711 [queue] [@server]

4712 where:

4713 *queue* The name of a valid execution or routing queue at the batch server denoted by
 4714 @*server*, defined as a string of up to 15 alphanumeric characters in the portable
 4715 character set (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 6.1,
 4716 Portable Character Set) where the first character is alphabetic.

4717 *server* The name of a batch server, defined as a string of alphanumeric characters in the
 4718 portable character set.

4719 If the application omits the batch *server* portion of a destination, then the utility shall use either:

- 4720 • The name of the default batch server
- 4721 or:
- 4722 • The name of the batch server that is currently managing the batch job

4723 The means by which a utility determines the default batch server is implementation-defined.

4724 If the application omits the *queue* portion of a destination, then the utility shall use the name of
 4725 the default queue at the batch server chosen. The means by which a batch server determines its
 4726 default queue is implementation-defined. If a destination is specified in the *queue@server* form,
 4727 then the utility shall use the specified queue at the specified server.

4728 A strictly conforming application shall use the syntax described for a destination. Whenever a
 4729 destination is specified whose syntax is not recognized by an implementation, then a message
 4730 shall be written to standard error and the utility shall exit with an exit status greater than zero.

4731 3.3.3 Multiple Keyword-Value Pairs

4732 For each option that can have multiple keyword-value pair arguments, the following rules shall
 4733 apply. Examples of options that can have list-oriented option-arguments are **-u** *value@keyword*
 4734 and **-l** *keyword=value*.

- 4735 1. If a batch utility is presented with a list-oriented option-argument for which a keyword has
 4736 a corresponding value that begins with a single or double quote, then the utility shall stop
 4737 interpreting the input stream for delimiters until a second single or double quote,
 4738 respectively, is encountered. This feature allows some flexibility for a comma (',') or
 4739 equals sign ('=') to be part of the value string for a particular keyword; for example:

4740 keywd1='val1,val2',keywd2="val3,val4"

4741 **Note:** This may require the user to escape the quotes as in the following command:

4742 foo -xkeywd1=\'val1,val2\',keywd2=\"val3,val4\"

- 4743 2. If a batch server is presented with a list-oriented attribute that has a keyword that was
 4744 encountered earlier in the list, then the later entry for that keyword shall replace the earlier
 4745 entry.
- 4746 3. If a batch server is presented with a list-oriented attribute that has a keyword without any
 4747 corresponding value of the form *keyword=* or *@keyword* and the same keyword was
 4748 encountered earlier in the list, then the prior entry for that keyword shall be ignored by the
 4749 batch server.
- 4750 4. If a batch utility is expecting a list-oriented option-argument entry of the form
 4751 *keyword=value*, but is presented with an entry of the form *keyword* without any
 4752 corresponding *value*, then the entry shall be treated as though a default value of NULL was
 4753 assigned (that is, *keyword=NULL*) for entry parsing purposes. The utility shall include only
 4754 the keyword, not the NULL value, in the associated job attribute.
- 4755 5. If a batch utility is expecting a list-oriented option-argument entry of the form
 4756 *value@keyword*, but is presented with an entry of the form *value* without any corresponding
 4757 *keyword*, then the entry shall be treated as though a keyword of NULL was assigned (that
 4758 is, *value@NULL*) for entry parsing purposes. The utility shall include only the value, not
 4759 the NULL keyword, in the associated job attribute.
- 4760 6. A batch server shall accept a list-oriented attribute that has multiple occurrences of the
 4761 same keyword, interpreting the keywords, in order, with the last value encountered taking
 4762 precedence over prior instances of the same keyword. This rule allows, but does not
 4763 require, a batch utility to preprocess the attribute to remove duplicate keywords.
- 4764 7. If a batch utility is presented with multiple list-oriented option-arguments on the
 4765 command line or in script directives, or both, for a single option, then the utility shall
 4766 concatenate, in order, any command line keyword and value pairs to the end of any
 4767 directive keyword and value pairs separated by a single comma to produce a single string
 4768 that is an equivalent, valid option-argument. The resulting string shall be assigned to the
 4769 associated attribute of the batch job (after optionally removing duplicate entries as
 4770 described in item 6).

Utilities

4771

4772

This chapter contains the definitions of the utilities, as follows:

4773

- Mandatory utilities that are present on every conformant system

4774

4775

4776

- Optional utilities that are present only on systems supporting the associated option; see Section 1.8.1 (on page 9) for information on the options in this volume of IEEE Std 1003.1-2001

4777 **NAME**4778 admin — create and administer SCCS files (**DEVELOPMENT**)4779 **SYNOPSIS**4780 xSI admin -i[name] [-n] [-a login] [-d flag] [-e login] [-f flag] [-m mrlist]
4781 [-r rel] [-t[name] [-y[comment]]] newfile4782 admin -n[-a login] [-d flag] [-e login] [-f flag] [-m mrlist] [-t[name]]
4783 [-y[comment]] newfile ...

4784 admin [-a login] [-d flag] [-m mrlist] [-r rel] [-t[name]] file ...

4785 admin -h file ...

4786 admin -z file ...

4787

4788 **DESCRIPTION**4789 The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named
4790 file does not exist, it shall be created, and its parameters shall be initialized according to the
4791 specified options. Parameters not initialized by an option shall be assigned a default value. If a
4792 named file does exist, parameters corresponding to specified options shall be changed, and other
4793 parameters shall be left as is.4794 All SCCS filenames supplied by the application shall be of the form *s.filename*. New SCCS files
4795 shall be given read-only permission mode. Write permission in the parent directory is required
4796 to create a file. All writing done by *admin* shall be to a temporary *x-file*, named *x.filename* (see *get*)
4797 created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode
4798 as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file
4799 shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This
4800 ensures that changes are made to the SCCS file only if no errors occur.4801 The *admin* utility shall also use a transient lock file (named *z.filename*), which is used to prevent
4802 simultaneous updates to the SCCS file; see *get*.4803 **OPTIONS**4804 The *admin* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section
4805 12.2, Utility Syntax Guidelines, except that the **-i**, **-t**, and **-y** options have optional option-
4806 arguments. These optional option-arguments shall not be presented as separate arguments. The
4807 following options are supported:4808 **-n** Create a new SCCS file. When **-n** is used without **-i**, the SCCS file shall be created
4809 with control information but without any file data.4810 **-i[name]** Specify the *name* of a file from which the text for a new SCCS file shall be taken.
4811 The text constitutes the first delta of the file (see the **-r** option for the delta
4812 numbering scheme). If the **-i** option is used, but the *name* option-argument is
4813 omitted, the text shall be obtained by reading the standard input. If this option is
4814 omitted, the SCCS file shall be created with control information but without any
4815 file data. The **-i** option implies the **-n** option.4816 **-r SID** Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that
4817 is, the branch and sequence numbers shall be zero or missing. The level number is
4818 optional, and defaults to 1.4819 **-t[name]** Specify the *name* of a file from which descriptive text for the SCCS file shall be
4820 taken. In the case of existing SCCS files (neither **-i** nor **-n** is specified):

4821		• A -t option without a <i>name</i> option-argument shall cause the removal of
4822		descriptive text (if any) currently in the SCCS file.
4823		• A -t option with a <i>name</i> option-argument shall cause the text (if any) in the
4824		named file to replace the descriptive text (if any) currently in the SCCS file.
4825	-f flag	Specify a <i>flag</i> , and, possibly, a value for the <i>flag</i> , to be placed in the SCCS file.
4826		Several -f options may be supplied on a single <i>admin</i> command line.
4827		Implementations shall recognize the following flags and associated values:
4828	b	Allow use of the -b option on a <i>get</i> command to create branch deltas.
4829	cceil	Specify the highest release (that is, ceiling), a number less than or equal to
4830		9999, which may be retrieved by a <i>get</i> command for editing. The default
4831		value for an unspecified c flag shall be 9999.
4832	ffloor	Specify the lowest release (that is, floor), a number greater than 0 but less
4833		than 9999, which may be retrieved by a <i>get</i> command for editing. The
4834		default value for an unspecified f flag shall be 1.
4835	dSID	Specify the default delta number (SID) to be used by a <i>get</i> command.
4836	istr	Treat the “No ID keywords” message issued by <i>get</i> or <i>delta</i> as a fatal
4837		error. In the absence of this flag, the message is only a warning. The
4838		message is issued if no SCCS identification keywords (see <i>get</i>) are found
4839		in the text retrieved or stored in the SCCS file. If a value is supplied, the
4840		application shall ensure that the keywords exactly match the given string;
4841		however, the string shall contain a keyword, and no embedded
4842		<newline>s.
4843	j	Allow concurrent <i>get</i> commands for editing on the same SID of an SCCS
4844		file. This allows multiple concurrent updates to the same version of the
4845		SCCS file.
4846	l list	Specify a <i>list</i> of releases to which deltas can no longer be made (that is, <i>get</i>
4847		-e against one of these locked releases fails). Conforming applications
4848		shall use the following syntax to specify a <i>list</i> . Implementations may
4849		accept additional forms as an extension:
4850		<list> ::= a <range-list>
4851		<range-list> ::= <range> <range-list>, <range>
4852		<range> ::= <SID>
4853		The character <i>a</i> in the <i>list</i> shall be equivalent to specifying all releases for
4854		the named SCCS file. The non-terminal <SID> in range shall be the delta
4855		number of an existing delta associated with the SCCS file.
4856	n	Cause <i>delta</i> to create a null delta in each of those releases (if any) being
4857		skipped when a delta is made in a new release (for example, in making
4858		delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas
4859		shall serve as anchor points so that branch deltas may later be created
4860		from them. The absence of this flag shall cause skipped releases to be
4861		nonexistent in the SCCS file, preventing branch deltas from being created
4862		from them in the future. During the initial creation of an SCCS file, the n
4863		flag may be ignored; that is, if the -r option is used to set the release
4864		number of the initial SID to a value greater than 1, null deltas need not be
4865		created for the “skipped” releases.

4866	qtext	Substitute user-definable <i>text</i> for all occurrences of the %Q% keyword in the SCCS file text retrieved by <i>get</i> .
4867		
4868	mmod	Specify the module name of the SCCS file substituted for all occurrences of the %M% keyword in the SCCS file text retrieved by <i>get</i> . If the m flag is not specified, the value assigned shall be the name of the SCCS file with the leading ' . ' removed.
4869		
4870		
4871		
4872	ttype	Specify the <i>type</i> of module in the SCCS file substituted for all occurrences of the %Y% keyword in the SCCS file text retrieved by <i>get</i> .
4873		
4874	vpgm	Cause <i>delta</i> to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the application shall ensure that the m option is also used even if its value is null.)
4875		
4876		
4877		
4878		
4879	-d flag	Remove (delete) the specified <i>flag</i> from an SCCS file. Several -d options may be supplied on a single <i>admin</i> command. See the -f option for allowable <i>flag</i> names. (The l <i>list</i> flag gives a <i>list</i> of releases to be unlocked. See the -f option for further description of the l flag and the syntax of a <i>list</i> .)
4880		
4881		
4882		
4883	-a login	Specify a <i>login</i> name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID shall be equivalent to specifying all <i>login</i> names common to that group ID. Several -a options may be used on a single <i>admin</i> command line. As many <i>logins</i> , or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If <i>login</i> or group ID is preceded by a ' ! ', the users so specified shall be denied permission to make deltas.
4884		
4885		
4886		
4887		
4888		
4889		
4890	-e login	Specify a <i>login</i> name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several -e options may be used on a single <i>admin</i> command line.
4891		
4892		
4893		
4894	-y[comment]	Insert the <i>comment</i> text into the SCCS file as a comment for the initial delta in a manner identical to that of <i>delta</i> . In the POSIX locale, omission of the -y option shall result in a default comment line being inserted in the form:
4895		
4896		
4897		"date and time created %s %s by %s", <date>, <time>, <login>
4898		where <date> is expressed in the format of the <i>date</i> utility's %Y/%m/%d conversion specification, <time> in the format of the <i>date</i> utility's %T conversion specification format, and <login> is the login name of the user creating the file.
4899		
4900		
4901	-m mrlist	Insert the list of modification request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to <i>delta</i> . The application shall ensure that the v flag is set and the MR numbers are validated if the v flag has a value (the name of an MR number validation program). A diagnostic message shall be written if the v flag is not set or MR validation fails.
4902		
4903		
4904		
4905		
4906	-h	Check the structure of the SCCS file and compare the newly computed checksum with the checksum that is stored in the SCCS file. If the newly computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.
4907		
4908		
4909		
4910	-z	Recompute the SCCS file checksum and store it in the first line of the SCCS file (see the -h option above). Note that use of this option on a truly corrupted file may
4911		

4912 prevent future detection of the corruption.

4913 OPERANDS

4914 The following operands shall be supported:

4915 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin*
 4916 utility shall behave as though each file in the directory were specified as a named
 4917 file, except that non-SCCS files (last component of the pathname does not begin
 4918 with **s.**) and unreadable files shall be silently ignored.

4919 *newfile* A pathname of an SCCS file to be created.

4920 If exactly one *file* or *newfile* operand appears, and it is **'-'**, the standard input shall be read; each
 4921 line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-
 4922 SCCS files and unreadable files shall be silently ignored.

4923 STDIN

4924 The standard input shall be a text file used only if **-i** is specified without an option-argument or
 4925 if a *file* or *newfile* operand is specified as **'-'**. If the first character of any standard input line is
 4926 <SOH> in the POSIX locale, the results are unspecified.

4927 INPUT FILES

4928 The existing SCCS files shall be text files of an unspecified format.

4929 The application shall ensure that the file named by the **-i** option's *name* option-argument shall be
 4930 a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results are
 4931 unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the
 4932 header for this file shall be 99 999 for this delta.

4933 ENVIRONMENT VARIABLES

4934 The following environment variables shall affect the execution of *admin*:

4935 *LANG* Provide a default value for the internationalization variables that are unset or null.
 4936 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,
 4937 Internationalization Variables for the precedence of internationalization variables
 4938 used to determine the values of locale categories.)

4939 *LC_ALL* If set to a non-empty string value, override the values of all the other
 4940 internationalization variables.

4941 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 4942 characters (for example, single-byte as opposed to multi-byte characters in
 4943 arguments and input files).

4944 *LC_MESSAGES*
 4945 Determine the locale that should be used to affect the format and contents of
 4946 diagnostic messages written to standard error and the contents of the default **-y**
 4947 comment.

4948 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

4949 ASYNCHRONOUS EVENTS

4950 Default.

4951 STDOUT

4952 Not used.

4953 STDERR

4954 The standard error shall be used only for diagnostic messages.

4955 OUTPUT FILES

4956 Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a
4957 locking *z-file*, as described in *get* (on page 476), may be created and deleted.

4958 EXTENDED DESCRIPTION

4959 None.

4960 EXIT STATUS

4961 The following exit values shall be returned:

4962 0 Successful completion.

4963 >0 An error occurred.

4964 CONSEQUENCES OF ERRORS

4965 Default.

4966 APPLICATION USAGE

4967 It is recommended that directories containing SCCS files be writable by the owner only, and that
4968 SCCS files themselves be read-only. The mode of the directories should allow only the owner to
4969 modify SCCS files contained in the directories. The mode of the SCCS files prevents any
4970 modification at all except by SCCS commands.

4971 EXAMPLES

4972 None.

4973 RATIONALE

4974 None.

4975 FUTURE DIRECTIONS

4976 None.

4977 SEE ALSO

4978 *delta, get, prs, what*

4979 CHANGE HISTORY

4980 First released in Issue 2.

4981 Issue 6

4982 The normative text is reworded to avoid use of the term “must” for application requirements.

4983 The normative text is reworded to emphasize the term “shall” for implementation requirements.

4984 The grammar is updated.

4985 The Open Group Base Resolution bwg2001-007 is applied, adding new text to the INPUT FILES
4986 section warning that the maximum lines recorded in the file is 99 999.

4987 The Open Group Base Resolution bwg2001-009 is applied, amending the description of the **-h**
4988 option.

4989 **NAME**

4990 alias — define or display aliases

4991 **SYNOPSIS**4992 UP alias [*alias-name*[=*string*] ...]

4993

4994 **DESCRIPTION**

4995 The *alias* utility shall create or redefine alias definitions or write the values of existing alias
 4996 definitions to standard output. An alias definition provides a string value that shall replace a
 4997 command name when it is encountered; see Section 2.3.1 (on page 32).

4998 An alias definition shall affect the current shell execution environment and the execution
 4999 environments of the subshells of the current shell. When used as specified by this volume of
 5000 IEEE Std 1003.1-2001, the alias definition shall not affect the parent process of the current shell
 5001 nor any utility environment invoked by the shell; see Section 2.12 (on page 61).

5002 **OPTIONS**

5003 None.

5004 **OPERANDS**

5005 The following operands shall be supported:

5006 *alias-name* Write the alias definition to standard output.5007 *alias-name=string*5008 Assign the value of *string* to the alias *alias-name*.

5009 If no operands are given, all alias definitions shall be written to standard output.

5010 **STDIN**

5011 Not used.

5012 **INPUT FILES**

5013 None.

5014 **ENVIRONMENT VARIABLES**5015 The following environment variables shall affect the execution of *alias*:

5016 *LANG* Provide a default value for the internationalization variables that are unset or null.
 5017 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,
 5018 Internationalization Variables for the precedence of internationalization variables
 5019 used to determine the values of locale categories.)

5020 *LC_ALL* If set to a non-empty string value, override the values of all the other
 5021 internationalization variables.

5022 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 5023 characters (for example, single-byte as opposed to multi-byte characters in
 5024 arguments).

5025 *LC_MESSAGES*

5026 Determine the locale that should be used to affect the format and contents of
 5027 diagnostic messages written to standard error.

5028 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

5029 **ASYNCHRONOUS EVENTS**

5030 Default.

5031 **STDOUT**5032 The format for displaying aliases (when no operands or only *name* operands are specified) shall
5033 be:5034 "%s=%s\n", *name*, *value*5035 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the
5036 shell. See the description of shell quoting in Section 2.2 (on page 30).5037 **STDERR**

5038 The standard error shall be used only for diagnostic messages.

5039 **OUTPUT FILES**

5040 None.

5041 **EXTENDED DESCRIPTION**

5042 None.

5043 **EXIT STATUS**

5044 The following exit values shall be returned:

5045 0 Successful completion.

5046 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.5047 **CONSEQUENCES OF ERRORS**

5048 Default.

5049 **APPLICATION USAGE**

5050 None.

5051 **EXAMPLES**5052 1. Change *ls* to give a columnated, more annotated output:5053 `alias ls="ls -CF"`

5054 2. Create a simple “redo” command to repeat previous entries in the command history file:

5055 `alias r='fc -s'`5056 3. Use 1K units for *du*:5057 `alias du=du\ -k`5058 4. Set up *nohup* so that it can deal with an argument that is itself an alias name:5059 `alias nohup="nohup "`5060 **RATIONALE**5061 The *alias* description is based on historical KornShell implementations. Known differences exist
5062 between that and the C shell. The KornShell version was adopted to be consistent with all the
5063 other KornShell features in this volume of IEEE Std 1003.1-2001, such as command line editing.5064 Since *alias* affects the current shell execution environment, it is generally provided as a shell
5065 regular built-in.5066 Historical versions of the KornShell have allowed aliases to be exported to scripts that are
5067 invoked by the same shell. This is triggered by the *alias -x* flag; it is allowed by this volume of
5068 IEEE Std 1003.1-2001 only when an explicit extension such as *-x* is used. The standard
5069 developers considered that aliases were of use primarily to interactive users and that they

5070 should normally not affect shell scripts called by those users; functions are available to such
5071 scripts.

5072 Historical versions of the KornShell had not written aliases in a quoted manner suitable for
5073 reentry to the shell, but this volume of IEEE Std 1003.1-2001 has made this a requirement for all
5074 similar output. Therefore, consistency with this volume of IEEE Std 1003.1-2001 was chosen over
5075 this detail of historical practice.

5076 **FUTURE DIRECTIONS**

5077 None.

5078 **SEE ALSO**

5079 Section 2.9.5 (on page 54)

5080 **CHANGE HISTORY**

5081 First released in Issue 4.

5082 **Issue 6**

5083 This utility is marked as part of the User Portability Utilities option.

5084 The APPLICATION USAGE section is added.

5085 **NAME**

5086 ar — create and maintain library archives

5087 **SYNOPSIS**5088 SD ar -d[-v] *archive file ...*

5089

5090 XSI ar -m [-v] *archive file ...* |5091 ar -m -a[-v] *posname archive file ...* |5092 ar -m -b[-v] *posname archive file ...* |5093 ar -m -i[-v] *posname archive file ...* |

5094

5095 XSI ar -p[-v] [-s] *archive [file ...]*5096 XSI ar -q[-cv] *archive file ...*

5097

5098 ar -r[-cuv] *archive file ...* |5099 XSI ar -r -a[-cuv] *posname archive file ...* |5100 ar -r -b[-cuv] *posname archive file ...* |5101 ar -r -i[-cuv] *posname archive file ...* |

5102

5103 XSI ar -t[-v] [-s] *archive [file ...]* |5104 XSI ar -x[-v] [-sCT] *archive [file ...]*5105 **DESCRIPTION**5106 The *ar* utility is part of the Software Development Utilities option.

5107 The *ar* utility can be used to create and maintain groups of files combined into an archive. Once
 5108 an archive has been created, new files can be added, and existing files in an archive can be
 5109 extracted, deleted, or replaced. When an archive consists entirely of valid object files, the
 5110 implementation shall format the archive so that it is usable as a library for link editing (see *c99*
 5111 and *fort77*). When some of the archived files are not valid object files, the suitability of the
 5112 XSI archive for library use is undefined. If an archive consists entirely of printable files, the entire
 5113 archive shall be printable.

5114 When *ar* creates an archive, it creates administrative information indicating whether a symbol
 5115 table is present in the archive. When there is at least one object file that *ar* recognizes as such in
 5116 the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is
 5117 used by the link editor to search the archive. Whenever the *ar* utility is used to create or update
 5118 the contents of such an archive, the symbol table shall be rebuilt. The *-s* option shall force the
 5119 symbol table to be rebuilt.

5120 All *file* operands can be pathnames. However, files within archives shall be named by a filename,
 5121 which is the last component of the pathname used when the file was entered into the archive.
 5122 The comparison of *file* operands to the names of files in archives shall be performed by
 5123 comparing the last component of the operand to the name of the file in the archive.

5124 It is unspecified whether multiple files in the archive may be identically named. In the case of
 5125 XSI such files, however, each *file* and *posname* operand shall match only the first file in the archive
 5126 having a name that is the same as the last component of the operand.

5127 **OPTIONS**

5128 The *ar* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,
 5129 Utility Syntax Guidelines.

5130 The following options shall be supported:

- 5131 XSI **-a** Position new files in the archive after the file named by the *posname* operand.
- 5132 XSI **-b** Position new files in the archive before the file named by the *posname* operand.
- 5133 **-c** Suppress the diagnostic message that is written to standard error by default when
 5134 the archive *archive* is created.
- 5135 XSI **-C** Prevent extracted files from replacing like-named files in the file system. This
 5136 option is useful when **-T** is also used, to prevent truncated filenames from
 5137 replacing files with the same prefix.
- 5138 **-d** Delete one or more *files* from *archive*.
- 5139 XSI **-i** Position new files in the archive before the file in the archive named by the *posname*
 5140 operand (equivalent to **-b**).
- 5141 XSI **-m** Move the named files in the archive. The **-a**, **-b**, or **-i** options with the *posname*
 5142 operand indicate the position; otherwise, move the names files in the archive to the
 5143 end of the archive.
- 5144 **-p** Write the contents of the *files* in the archive named by *file* operands from *archive* to
 5145 the standard output. If no *file* operands are specified, the contents of all files in the
 5146 archive shall be written in the order of the archive.
- 5147 XSI **-q** Append the named files to the end of the archive. In this case *ar* does not check
 5148 whether the added files are already in the archive. This is useful to bypass the
 5149 searching otherwise done when creating a large archive piece by piece.
- 5150 **-r** Replace or add *files* to *archive*. If the archive named by *archive* does not exist, a
 5151 new archive shall be created and a diagnostic message shall be written to standard
 5152 error (unless the **-c** option is specified). If no *files* are specified and the *archive*
 5153 exists, the results are undefined. Files that replace existing files in the archive shall
 5154 not change the order of the archive. Files that do not replace existing files in the
 5155 XSI archive shall be appended to the archive unless a **-a**, **-b**, or **-i** option specifies
 5156 another position.
- 5157 XSI **-s** Force the regeneration of the archive symbol table even if *ar* is not invoked with an
 5158 option that modifies the archive contents. This option is useful to restore the
 5159 archive symbol table after it has been stripped; see *strip*.
- 5160 **-t** Write a table of contents of *archive* to the standard output. The files specified by the
 5161 *file* operands shall be included in the written list. If no *file* operands are specified,
 5162 all files in *archive* shall be included in the order of the archive.
- 5163 XSI **-T** Allow filename truncation of extracted files whose archive names are longer than
 5164 the file system can support. By default, extracting a file with a name that is too
 5165 long shall be an error; a diagnostic message shall be written and the file shall not
 5166 be extracted.
- 5167 **-u** Update older files in the archive. When used with the **-r** option, files in the archive
 5168 shall be replaced only if the corresponding *file* has a modification time that is at
 5169 least as new as the modification time of the file in the archive.

5170 **-v** Give verbose output. When used with the option characters **-d**, **-r**, or **-x**, write a
 5171 detailed file-by-file description of the archive creation and maintenance activity, as
 5172 described in the STDOUT section.

5173 When used with **-p**, write the name of the file in the archive to the standard output
 5174 before writing the file in the archive itself to the standard output, as described in
 5175 the STDOUT section.

5176 When used with **-t**, include a long listing of information about the files in the
 5177 archive, as described in the STDOUT section.

5178 **-x** Extract the files in the archive named by the *file* operands from *archive*. The
 5179 contents of the archive shall not be changed. If no *file* operands are given, all files
 5180 in the archive shall be extracted. The modification time of each file extracted shall
 5181 be set to the time the file is extracted from the archive.

5182 OPERANDS

5183 The following operands shall be supported:

5184 *archive* A pathname of the archive.

5185 *file* A pathname. Only the last component shall be used when comparing against the
 5186 names of files in the archive. If two or more *file* operands have the same last
 5187 pathname component (basename), the results are unspecified. The
 5188 implementation's archive format shall not truncate valid filenames of files added
 5189 to or replaced in the archive.

5190 xSI *posname* The name of a file in the archive, used for relative positioning; see options **-m** and
 5191 **-r**.

5192 STDIN

5193 Not used.

5194 INPUT FILES

5195 The archive named by *archive* shall be a file in the format created by *ar -r*.

5196 ENVIRONMENT VARIABLES

5197 The following environment variables shall affect the execution of *ar*:

5198 *LANG* Provide a default value for the internationalization variables that are unset or null.
 5199 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,
 5200 Internationalization Variables for the precedence of internationalization variables
 5201 used to determine the values of locale categories.)

5202 *LC_ALL* If set to a non-empty string value, override the values of all the other
 5203 internationalization variables.

5204 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 5205 characters (for example, single-byte as opposed to multi-byte characters in
 5206 arguments and input files).

5207 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
 5208 diagnostic messages written to standard error.
 5209

5210 *LC_TIME* Determine the format and content for date and time strings written by *ar -tv*.

5211 xSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

5212 *TMPDIR* Determine the pathname that overrides the default directory for temporary files, if
 5213 any.

5214 *TZ* Determine the timezone used to calculate date and time strings written by *ar -tv*.
 5215 If *TZ* is unset or null, an unspecified default timezone shall be used.

5216 **ASYNCHRONOUS EVENTS**

5217 Default.

5218 **STDOUT**

5219 If the **-d** option is used with the **-v** option, the standard output format shall be:

5220 "d - %s\n", <file>

5221 where *file* is the operand specified on the command line.

5222 If the **-p** option is used with the **-v** option, *ar* shall precede the contents of each file with:

5223 "\n<%s>\n\n", <file>

5224 where *file* is the operand specified on the command line, if *file* operands were specified, and the
 5225 name of the file in the archive if they were not.

5226 If the **-r** option is used with the **-v** option:

- 5227 • If *file* is already in the archive, the standard output format shall be:

5228 "r - %s\n", <file>

5229 where <*file*> is the operand specified on the command line.

- 5230 • If *file* is not already in the archive, the standard output format shall be:

5231 "a - %s\n", <file>

5232 where <*file*> is the operand specified on the command line.

5233 If the **-t** option is used, *ar* shall write the names of the files in the archive to the standard output
 5234 in the format:

5235 "%s\n", <file>

5236 where *file* is the operand specified on the command line, if *file* operands were specified, or the
 5237 name of the file in the archive if they were not.

5238 If the **-t** option is used with the **-v** option, the standard output format shall be:

5239 "%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>,
 5240 <group ID>, <number of bytes in member>,
 5241 <abbreviated month>, <day-of-month>, <hour>,
 5242 <minute>, <year>, <file>

5243 where:

5244 <*file*> Shall be the operand specified on the command line, if *file* operands were specified,
 5245 or the name of the file in the archive if they were not.

5246 <*member mode*>

5247 Shall be formatted the same as the <*file mode*> string defined in the STDOUT
 5248 section of *ls*, except that the first character, the <*entry type*>, is not used; the string
 5249 represents the file mode of the file in the archive at the time it was added to or
 5250 replaced in the archive.

5251 The following represent the last-modification time of a file when it was most recently added to
 5252 or replaced in the archive:

5253 <*abbreviated month*>
 5254 Equivalent to the format of the %b conversion specification format in *date*.

5255 <*day-of-month*>
 5256 Equivalent to the format of the %e conversion specification format in *date*.

5257 <*hour*> Equivalent to the format of the %H conversion specification format in *date*.

5258 <*minute*> Equivalent to the format of the %M conversion specification format in *date*.

5259 <*year*> Equivalent to the format of the %Y conversion specification format in *date*.

5260 When *LC_TIME* does not specify the POSIX locale, a different format and order of presentation
 5261 of these fields relative to each other may be used in a format appropriate in the specified locale.

5262 If the **-x** option is used with the **-v** option, the standard output format shall be:

5263 "x - %s\n", <*file*>

5264 where *file* is the operand specified on the command line, if *file* operands were specified, or the
 5265 name of the file in the archive if they were not.

5266 **STDERR**
 5267 The standard error shall be used only for diagnostic messages. The diagnostic message about
 5268 creating a new archive when **-c** is not specified shall not modify the exit status.

5269 **OUTPUT FILES**
 5270 Archives are files with unspecified formats.

5271 **EXTENDED DESCRIPTION**
 5272 None.

5273 **EXIT STATUS**
 5274 The following exit values shall be returned:
 5275 0 Successful completion.
 5276 >0 An error occurred.

5277 **CONSEQUENCES OF ERRORS**
 5278 Default.

5279 **APPLICATION USAGE**
 5280 None.

5281 **EXAMPLES**
 5282 None.

5283 **RATIONALE**
 5284 The archive format is not described. It is recognized that there are several known *ar* formats,
 5285 which are not compatible. The *ar* utility is included, however, to allow creation of archives that
 5286 are intended for use only on one machine. The archive is specified as a file, and it can be moved
 5287 as a file. This does allow an archive to be moved from one machine to another machine that uses
 5288 the same implementation of *ar*.

5289 Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable “archives”. This is a not
 5290 a duplication; the *ar* utility is included to provide an interface primarily for *make* and the
 5291 compilers, based on a historical model.

5292 In historical implementations, the **-q** option (available on XSI-conforming systems) is known to
 5293 execute quickly because *ar* does not check on whether the added members are already in the
 5294 archive. This is useful to bypass the searching otherwise done when creating a large archive

5295 piece-by-piece. These remarks may but need not remain true for a brand new implementation of
5296 this utility; hence, these remarks have been moved into the RATIONALE.

5297 BSD implementations historically required applications to provide the **-s** option whenever the
5298 archive was supposed to contain a symbol table. As in this volume of IEEE Std 1003.1-2001,
5299 System V historically creates or updates an archive symbol table whenever an object file is
5300 removed from, added to, or updated in the archive.

5301 The OPERANDS section requires what might seem to be true without specifying it: the archive
5302 cannot truncate the filenames below {NAME_MAX}. Some historical implementations do so,
5303 however, causing unexpected results for the application. Therefore, this volume of
5304 IEEE Std 1003.1-2001 makes the requirement explicit to avoid misunderstandings.

5305 According to the System V documentation, the options **-dmpqrtx** are not required to begin with
5306 a hyphen ('-'). This volume of IEEE Std 1003.1-2001 requires that a conforming application use
5307 the leading hyphen.

5308 The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as
5309 an example:

5310 A file created by *ar* begins with the “magic” string `!<arch>\n`. The rest of the archive is
5311 made up of objects, each of which is composed of a header for a file, a possible filename, and
5312 the file contents. The header is portable between machine architectures, and, if the file
5313 contents are printable, the archive is itself printable.

5314 The header is made up of six ASCII fields, followed by a two-character trailer. The fields are
5315 the object name (16 characters), the file last modification time (12 characters), the user and
5316 group IDs (each 6 characters), the file mode (8 characters), and the file size (10 characters). All
5317 numeric fields are in decimal, except for the file mode, which is in octal.

5318 The modification time is the file `st_mtime` field. The user and group IDs are the file `st_uid` and
5319 `st_gid` fields. The file mode is the file `st_mode` field. The file size is the file `st_size` field. The
5320 two-byte trailer is the string `"\n"`.

5321 Only the name field has any provision for overflow. If any filename is more than 16
5322 characters in length or contains an embedded space, the string `#1/` followed by the ASCII
5323 length of the name is written in the name field. The file size (stored in the archive header) is
5324 incremented by the length of the name. The name is then written immediately following the
5325 archive header.

5326 Any unused characters in any of these fields are written as `<space>s`. If any fields are their
5327 particular maximum number of characters in length, there is no separation between the
5328 fields.

5329 Objects in the archive are always an even number of bytes long; files that are an odd number
5330 of bytes long are padded with a `<newline>`, although the size in the header does not reflect
5331 this.

5332 The *ar* utility description requires that (when all its members are valid object files) *ar* produce an
5333 object code library, which the linkage editor can use to extract object modules. If the linkage
5334 editor needs a symbol table to permit random access to the archive, *ar* must provide it; however,
5335 *ar* does not require a symbol table.

5336 The BSD **-o** option was omitted. It is a rare conforming application that uses *ar* to extract object
5337 code from a library with concern for its modification time, since this can only be of importance
5338 to *make*. Hence, since this functionality is not deemed important for applications portability, the
5339 modification time of the extracted files is set to the current time.

5340 There is at least one known implementation (for a small computer) that can accommodate only
 5341 object files for that system, disallowing mixed object and other files. The ability to handle any
 5342 type of file is not only historical practice for most implementations, but is also a reasonable
 5343 expectation.

5344 Consideration was given to changing the output format of *ar -tv* to the same format as the
 5345 output of *ls -l*. This would have made parsing the output of *ar* the same as that of *ls*. This was
 5346 rejected in part because the current *ar* format is commonly used and changes would break
 5347 historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a
 5348 slash. Changing this to be the user name and group name would not be correct if the archive
 5349 were moved to a machine that contained a different user database. Since *ar* cannot know
 5350 whether the archive was generated on the same machine, it cannot tell what to report.

5351 The text on the *-ur* option combination is historical practice—since one filename can easily
 5352 represent two different files (for example, */a/foo* and */b/foo*), it is reasonable to replace the file in
 5353 the archive even when the modification time in the archive is identical to that in the file system.

5354 FUTURE DIRECTIONS

5355 None.

5356 SEE ALSO

5357 *c99*, *date*, *fort77*, *pax*, *strip* the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13,
 5358 Headers, *<unistd.h>* description of {POSIX_NO_TRUNC}

5359 CHANGE HISTORY

5360 First released in Issue 2.

5361 Issue 5

5362 The FUTURE DIRECTIONS section is added.

5363 Issue 6

5364 This utility is marked as part of the Software Development Utilities option.

5365 The STDOUT description is changed for the *-v* option to align with the IEEE P1003.2b draft
 5366 standard.

5367 The normative text is reworded to avoid use of the term “must” for application requirements.

5368 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

5369 IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use
 5370 “file” to refer to a file in the file system hierarchy, “archive” to refer to the archive being
 5371 operated upon by the *ar* utility, and “file in the archive” to refer to a copy of a file that is
 5372 contained in the archive.

5373 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/10 is applied, making corrections to the
 5374 SYNOPSIS. The change was needed since the *-a*, *-b*, and *-i* options are mutually-exclusive, and
 5375 *posname* is required if any of these options is specified.

5376 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/11 is applied, correcting the description
 5377 of the two-byte trailer in RATIONALE which had missed out a backquote. The correct trailer is a
 5378 backquote followed by a <newline>.

5379 **NAME**

5380 asa — interpret carriage-control characters

5381 **SYNOPSIS**5382 FR asa [*file* ...]

5383

5384 **DESCRIPTION**5385 The *asa* utility shall write its input files to standard output, mapping carriage-control characters
5386 from the text files to line-printer control sequences in an implementation-defined manner.5387 The first character of every line shall be removed from the input, and the following actions are
5388 performed.

5389 If the character removed is:

5390 <space> The rest of the line is output without change.

5391 0 A <newline> is output, then the rest of the input line.

5392 1 One or more implementation-defined characters that causes an advance to the next
5393 page shall be output, followed by the rest of the input line.5394 + The <newline> of the previous line shall be replaced with one or more
5395 implementation-defined characters that causes printing to return to column position 1,
5396 followed by the rest of the input line. If the ' + ' is the first character in the input, it shall
5397 be equivalent to <space>.5398 The action of the *asa* utility is unspecified upon encountering any character other than those
5399 listed above as the first character in a line.5400 **OPTIONS**

5401 None.

5402 **OPERANDS**5403 *file* A pathname of a text file used for input. If no *file* operands are specified, the
5404 standard input shall be used.5405 **STDIN**5406 The standard input shall be used only if no *file* operands are specified; see the INPUT FILES
5407 section.5408 **INPUT FILES**

5409 The input files shall be text files.

5410 **ENVIRONMENT VARIABLES**5411 The following environment variables shall affect the execution of *asa*:5412 *LANG* Provide a default value for the internationalization variables that are unset or null.
5413 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,
5414 Internationalization Variables for the precedence of internationalization variables
5415 used to determine the values of locale categories.)5416 *LC_ALL* If set to a non-empty string value, override the values of all the other
5417 internationalization variables.5418 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
5419 characters (for example, single-byte as opposed to multi-byte characters in
5420 arguments and input files).

5421 *LC_MESSAGES*
 5422 Determine the locale that should be used to affect the format and contents of
 5423 diagnostic messages written to standard error.

5424 xSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

5425 **ASYNCHRONOUS EVENTS**
 5426 Default.

5427 **STDOUT**
 5428 The standard output shall be the text from the input file modified as described in the
 5429 DESCRIPTION section.

5430 **STDERR**
 5431 None.

5432 **OUTPUT FILES**
 5433 None.

5434 **EXTENDED DESCRIPTION**
 5435 None.

5436 **EXIT STATUS**
 5437 The following exit values shall be returned:
 5438 0 All input files were output successfully.
 5439 >0 An error occurred.

5440 **CONSEQUENCES OF ERRORS**
 5441 Default.

5442 **APPLICATION USAGE**
 5443 None.

5444 **EXAMPLES**

5445 1. The following command:
 5446 *asa file*
 5447 permits the viewing of *file* (created by a program using FORTRAN-style carriage-control
 5448 characters) on a terminal.

5449 2. The following command:
 5450 *a.out | asa | lp*
 5451 formats the FORTRAN output of **a.out** and directs it to the printer.

5452 **RATIONALE**
 5453 The *asa* utility is needed to map “standard” FORTRAN 77 output into a form acceptable to
 5454 contemporary printers. Usually, *asa* is used to pipe data to the *lp* utility; see *lp*.

5455 This utility is generally used only by FORTRAN programs. The standard developers decided to
 5456 retain *asa* to avoid breaking the historical large base of FORTRAN applications that put
 5457 carriage-control characters in their output files. There is no requirement that a system have a
 5458 FORTRAN compiler in order to run applications that need *asa*.

5459 Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII
 5460 <carriage-return> in response to a '+'. It is suggested that implementations treat characters
 5461 other than 0, 1, and '+' as <space> in the absence of any compelling reason to do otherwise.
 5462 However, the action is listed here as “unspecified”, permitting an implementation to provide

5463 extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.

5464 **FUTURE DIRECTIONS**

5465 None.

5466 **SEE ALSO**

5467 *fort77, lp*

5468 **CHANGE HISTORY**

5469 First released in Issue 4.

5470 **Issue 6**

5471 This utility is marked as part of the FORTRAN Runtime Utilities option.

5472 The normative text is reworded to avoid use of the term “must” for application requirements.

5473 **NAME**

5474 at — execute commands at a later time

5475 **SYNOPSIS**5476 UP at [-m] [-f *file*] [-q *queuenam*e] -t *time_arg*5477 at [-m] [-f *file*] [-q *queuenam*e] *timespec* ...5478 at -r *at_job_id* ...5479 at -l -q *queuenam*e5480 at -l [*at_job_id* ...]

5481

5482 **DESCRIPTION**5483 The *at* utility shall read commands from standard input and group them together as an *at-job*, to
5484 be executed at a later time.5485 The *at-job* shall be executed in a separate invocation of the shell, running in a separate process
5486 group with no controlling terminal, except that the environment variables, current working
5487 directory, file creation mask, and other implementation-defined execution-time attributes in
5488 effect when the *at* utility is executed shall be retained and used when the *at-job* is executed.5489 When the *at-job* is submitted, the *at_job_id* and scheduled time shall be written to standard error.
5490 The *at_job_id* is an identifier that shall be a string consisting solely of alphanumeric characters
5491 and the period character. The *at_job_id* shall be assigned by the system when the job is scheduled
5492 such that it uniquely identifies a particular job.5493 User notification and the processing of the job's standard output and standard error are
5494 described under the **-m** option.5495 XSI Users shall be permitted to use *at* if their name appears in the file **/usr/lib/cron/at.allow**. If that
5496 file does not exist, the file **/usr/lib/cron/at.deny** shall be checked to determine whether the user
5497 shall be denied access to *at*. If neither file exists, only a process with the appropriate privileges
5498 shall be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be
5499 permitted. The **at.allow** and **at.deny** files shall consist of one user name per line.5500 **OPTIONS**5501 The *at* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,
5502 Utility Syntax Guidelines.

5503 The following options shall be supported:

5504 **-f file** Specify the pathname of a file to be used as the source of the *at-job*, instead of
5505 standard input.5506 **-l** (The letter ell.) Report all jobs scheduled for the invoking user if no *at_job_id*
5507 operands are specified. If *at_job_ids* are specified, report only information for these
5508 jobs. The output shall be written to standard output.5509 **-m** Send mail to the invoking user after the *at-job* has run, announcing its completion.
5510 Standard output and standard error produced by the *at-job* shall be mailed to the
5511 user as well, unless redirected elsewhere. Mail shall be sent even if the job
5512 produces no output.5513 If **-m** is not used, the job's standard output and standard error shall be provided to
5514 the user by means of mail, unless they are redirected elsewhere; if there is no such
5515 output to provide, the implementation need not notify the user of the job's
5516 completion.

5517 **-q** *queue**name*
 5518 Specify in which queue to schedule a job for submission. When used with the **-l**
 5519 option, limit the search to that particular queue. By default, at-jobs shall be
 5520 scheduled in queue *a*. In contrast, queue *b* shall be reserved for batch jobs; see
 5521 *batch*. The meanings of all other *queue**names* are implementation-defined. If **-q** is
 5522 specified along with either of the **-t** *time_arg* or *timespec* arguments, the results are
 5523 unspecified.

5524 **-r** Remove the jobs with the specified *at_job_id* operands that were previously
 5525 scheduled by the *at* utility.

5526 **-t** *time_arg* Submit the job to be run at the time specified by the *time* option-argument, which
 5527 the application shall ensure has the format as specified by the *touch -t time* utility.

5528 OPERANDS

5529 The following operands shall be supported:

5530 *at_job_id* The name reported by a previous invocation of the *at* utility at the time the job was
 5531 scheduled.

5532 *timespec* Submit the job to be run at the date and time specified. All of the *timespec* operands
 5533 are interpreted as if they were separated by <space>s and concatenated, and shall
 5534 be parsed as described in the grammar at the end of this section. The date and time
 5535 shall be interpreted as being in the timezone of the user (as determined by the *TZ*
 5536 variable), unless a timezone name appears as part of *time*, below.

5537 In the POSIX locale, the following describes the three parts of the time
 5538 specification string. All of the values from the *LC_TIME* categories in the POSIX
 5539 locale shall be recognized in a case-insensitive manner.

5540 *time* The time can be specified as one, two, or four digits. One-digit and
 5541 two-digit numbers shall be taken to be hours; four-digit numbers to
 5542 be hours and minutes. The time can alternatively be specified as two
 5543 numbers separated by a colon, meaning *hour:minute*. An AM/PM
 5544 indication (one of the values from the **am_pm** keywords in the
 5545 *LC_TIME* locale category) can follow the time; otherwise, a 24-hour
 5546 clock time shall be understood. A timezone name can also follow to
 5547 further qualify the time. The acceptable timezone names are
 5548 implementation-defined, except that they shall be case-insensitive
 5549 and the string **utc** is supported to indicate the time is in Coordinated
 5550 Universal Time. In the POSIX locale, the *time* field can also be one of
 5551 the following tokens:

5552 **midnight** Indicates the time 12:00 am (00:00).

5553 **noon** Indicates the time 12:00 pm.

5554 **now** Indicates the current day and time. Invoking *at <now>*
 5555 shall submit an at-job for potentially immediate
 5556 execution (that is, subject only to unspecified
 5557 scheduling delays).

5558 *date* An optional *date* can be specified as either a month name (one of the
 5559 values from the **mon** or **abmon** keywords in the *LC_TIME* locale
 5560 category) followed by a day number (and possibly year number
 5561 preceded by a comma), or a day of the week (one of the values from
 5562 the **day** or **abday** keywords in the *LC_TIME* locale category). In the
 5563 POSIX locale, two special days shall be recognized:

5564 **today** Indicates the current day.

5565 **tomorrow** Indicates the day following the current day.

5566 If no *date* is given, **today** shall be assumed if the given time is greater
5567 than the current time, and **tomorrow** shall be assumed if it is less. If
5568 the given month is less than the current month (and no year is given),
5569 next year shall be assumed.

5570 *increment* The optional *increment* shall be a number preceded by a plus sign
5571 (' + ') and suffixed by one of the following: **minutes, hours, days,**
5572 **weeks, months, or years.** (The singular forms shall also be
5573 accepted.) The keyword **next** shall be equivalent to an increment
5574 number of +1. For example, the following are equivalent commands:

5575 at 2pm + 1 week
5576 at 2pm next week

5577 The following grammar describes the precise format of *timespec* in the POSIX locale. The general
5578 conventions for this style of grammar are described in Section 1.10 (on page 19). This formal
5579 syntax shall take precedence over the preceding text syntax description. The longest possible
5580 token or delimiter shall be recognized at a given point. When used in a *timespec*, white space
5581 shall also delimit tokens.

5582 %token hr24clock_hr_min
5583 %token hr24clock_hour
5584 /*
5585 An hr24clock_hr_min is a one, two, or four-digit number. A one-digit
5586 or two-digit number constitutes an hr24clock_hour. An hr24clock_hour
5587 may be any of the single digits [0,9], or may be double digits, ranging
5588 from [00,23]. If an hr24clock_hr_min is a four-digit number, the
5589 first two digits shall be a valid hr24clock_hour, while the last two
5590 represent the number of minutes, from [00,59].
5591 */

5592 %token wallclock_hr_min
5593 %token wallclock_hour
5594 /*
5595 A wallclock_hr_min is a one, two-digit, or four-digit number.
5596 A one-digit or two-digit number constitutes a wallclock_hour.
5597 A wallclock_hour may be any of the single digits [1,9], or may
5598 be double digits, ranging from [01,12]. If a wallclock_hr_min
5599 is a four-digit number, the first two digits shall be a valid
5600 wallclock_hour, while the last two represent the number of
5601 minutes, from [00,59].
5602 */

5603 %token minute
5604 /*
5605 A minute is a one or two-digit number whose value can be [0,9]
5606 or [00,59].
5607 */

5608 %token day_number
5609 /*
5610 A day_number is a number in the range appropriate for the particular
5611 month and year specified by month_name and year_number, respectively.

```

5612         If no year_number is given, the current year is assumed if the given
5613         date and time are later this year. If no year_number is given and
5614         the date and time have already occurred this year and the month is
5615         not the current month, next year is the assumed year.
5616     */

5617     %token year_number
5618     /*
5619         A year_number is a four-digit number representing the year A.D., in
5620         which the at_job is to be run.
5621     */

5622     %token inc_number
5623     /*
5624         The inc_number is the number of times the succeeding increment
5625         period is to be added to the specified date and time.
5626     */

5627     %token timezone_name
5628     /*
5629         The name of an optional timezone suffix to the time field, in an
5630         implementation-defined format.
5631     */

5632     %token month_name
5633     /*
5634         One of the values from the mon or abmon keywords in the LC_TIME
5635         locale category.
5636     */

5637     %token day_of_week
5638     /*
5639         One of the values from the day or abday keywords in the LC_TIME
5640         locale category.
5641     */

5642     %token am_pm
5643     /*
5644         One of the values from the am_pm keyword in the LC_TIME locale
5645         category.
5646     */

5647     %start timespec
5648     %%
5649     timespec      : time
5650                   | time date
5651                   | time increment
5652                   | time date increment
5653                   | nowspec
5654                   ;
5655     nowspec       : "now"
5656                   | "now" increment
5657                   ;
5658     time          : hr24clock_hr_min
5659                   | hr24clock_hr_min timezone_name

```

```

5660         | hr24clock_hour ":" minute
5661         | hr24clock_hour ":" minute timezone_name
5662         | wallclock_hr_min am_pm
5663         | wallclock_hr_min am_pm timezone_name
5664         | wallclock_hour ":" minute am_pm
5665         | wallclock_hour ":" minute am_pm timezone_name
5666         | "noon"
5667         | "midnight"
5668         ;

5669     date      : month_name day_number
5670         | month_name day_number "," year_number
5671         | day_of_week
5672         | "today"
5673         | "tomorrow"
5674         ;

5675     increment  : "+" inc_number inc_period
5676         | "next" inc_period
5677         ;

5678     inc_period : "minute" | "minutes"
5679         | "hour" | "hours"
5680         | "day" | "days"
5681         | "week" | "weeks"
5682         | "month" | "months"
5683         | "year" | "years"
5684         ;

```

5685 STDIN

5686 The standard input shall be a text file consisting of commands acceptable to the shell command
 5687 language described in Chapter 2 (on page 29). The standard input shall only be used if no **-f file**
 5688 option is specified.

5689 INPUT FILES

5690 See the STDIN section.

5691 XSI The text files **/usr/lib/cron/at.allow** and **/usr/lib/cron/at.deny** shall contain zero or more user
 5692 names, one per line, of users who are, respectively, authorized or denied access to the *at* and
 5693 *batch* utilities.

5694 ENVIRONMENT VARIABLES

5695 The following environment variables shall affect the execution of *at*:

5696 **LANG** Provide a default value for the internationalization variables that are unset or null.
 5697 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,
 5698 Internationalization Variables for the precedence of internationalization variables
 5699 used to determine the values of locale categories.)

5700 **LC_ALL** If set to a non-empty string value, override the values of all the other
 5701 internationalization variables.

5702 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 5703 characters (for example, single-byte as opposed to multi-byte characters in
 5704 arguments and input files).

5705 **LC_MESSAGES**

5706 Determine the locale that should be used to affect the format and contents of

5707 diagnostic messages written to standard error and informative messages written to
 5708 standard output.

5709 XSI **NLS_PATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

5710 **LC_TIME** Determine the format and contents for date and time strings written and accepted
 5711 by *at*.

5712 **SHELL** Determine a name of a command interpreter to be used to invoke the *at*-job. If the
 5713 variable is unset or null, *sh* shall be used. If it is set to a value other than a name for
 5714 *sh*, the implementation shall do one of the following: use that shell; use *sh*; use the
 5715 login shell from the user database; or any of the preceding accompanied by a
 5716 warning diagnostic about which was chosen.

5717 **TZ** Determine the timezone. The job shall be submitted for execution at the time
 5718 specified by *timespec* or *-t time* relative to the timezone specified by the *TZ*
 5719 variable. If *timespec* specifies a timezone, it shall override *TZ*. If *timespec* does not
 5720 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall
 5721 be used.

5722 **ASYNCHRONOUS EVENTS**

5723 Default.

5724 **STDOUT**

5725 When standard input is a terminal, prompts of unspecified format for each line of the user input
 5726 described in the STDIN section may be written to standard output.

5727 In the POSIX locale, the following shall be written to the standard output for each job when jobs
 5728 are listed in response to the *-l* option:

5729 "%s\t%s\n", *at_job_id*, *<date>*

5730 where *date* shall be equivalent in format to the output of:

5731 date +"%a %b %e %T %Y"

5732 The date and time written shall be adjusted so that they appear in the timezone of the user (as
 5733 determined by the *TZ* variable).

5734 **STDERR**

5735 In the POSIX locale, the following shall be written to standard error when a job has been
 5736 successfully submitted:

5737 "job %s at %s\n", *at_job_id*, *<date>*

5738 where *date* has the same format as that described in the STDOUT section. Neither this, nor
 5739 warning messages concerning the selection of the command interpreter, shall be considered a
 5740 diagnostic that changes the exit status.

5741 Diagnostic messages, if any, shall be written to standard error.

5742 **OUTPUT FILES**

5743 None.

5744 **EXTENDED DESCRIPTION**

5745 None.

5746 **EXIT STATUS**

5747 The following exit values shall be returned:

5748 0 The *at* utility successfully submitted, removed, or listed a job or jobs.

5749 >0 An error occurred.

5750 CONSEQUENCES OF ERRORS

5751 The job shall not be scheduled, removed, or listed.

5752 APPLICATION USAGE

5753 The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other
5754 cultures may be supported with substantially different interfaces, although implementations are
5755 encouraged to provide comparable levels of functionality.

5756 Since the commands run in a separate shell invocation, running in a separate process group with
5757 no controlling terminal, open file descriptors, traps, and priority inherited from the invoking
5758 environment are lost.

5759 Some implementations do not allow substitution of different shells using *SHELL*. System V
5760 systems, for example, have used the login shell value for the user in */etc/passwd*. To select
5761 reliably another command interpreter, the user must include it as part of the script, such as:

```
5762       $ at 1800
5763       myshell myscript
5764       EOT
5765       job ... at ...
5766       $
```

5767 EXAMPLES

5768 1. This sequence can be used at a terminal:

```
5769           at -m 0730 tomorrow
5770           sort < file >outfile
5771           EOT
```

5772 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a
5773 command procedure (the sequence of output redirection specifications is significant):

```
5774           at now + 1 hour <<!
5775           diff file1 file2 2>&1 >outfile | mailx mygroup
5776           !
```

5777 3. To have a job reschedule itself, *at* can be invoked from within the *at*-job. For example, this
5778 daily processing script named **my.daily** runs every day (although *crontab* is a more
5779 appropriate vehicle for such work):

```
5780           # my.daily runs every day
5781           daily processing
5782           at now tomorrow < my.daily
```

5783 4. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as
5784 there are no ambiguities. Examples of various times and operand presentation include:

```
5785           at 0815am Jan 24
5786           at 8 :15amjan24
5787           at now "+ 1day"
5788           at 5 pm FRIday
5789           at '17
5790                utc+
5791                30minutes'
```

5792 **RATIONALE**

5793 The *at* utility reads from standard input the commands to be executed at a later time. It may be
 5794 useful to redirect standard output and standard error within the specified commands.

5795 The **-t** *time* option was added as a new capability to support an internationalized way of
 5796 specifying a time for execution of the submitted job.

5797 Early proposals added a “jobname” concept as a way of giving submitted jobs names that are
 5798 meaningful to the user submitting them. The historical, system-specified *at_job_id* gives no
 5799 indication of what the job is. Upon further reflection, it was decided that the benefit of this was
 5800 not worth the change in historical interface. The *at* functionality is useful in simple
 5801 environments, but in large or complex situations, the functionality provided by the Batch
 5802 Services option is more suitable.

5803 The **-q** option historically has been an undocumented option, used mainly by the *batch* utility.

5804 The System V **-m** option was added to provide a method for informing users that an at-job had
 5805 completed. Otherwise, users are only informed when output to standard error or standard
 5806 output are not redirected.

5807 The behavior of *at* **<now>** was changed in an early proposal from being unspecified to
 5808 submitting a job for potentially immediate execution. Historical BSD *at* implementations
 5809 support this. Historical System V implementations give an error in that case, but a change to the
 5810 System V versions should have no backwards-compatibility ramifications.

5811 On BSD-based systems, a **-u** *user* option has allowed those with appropriate privileges to access
 5812 the work of other users. Since this is primarily a system administration feature and is not
 5813 universally implemented, it has been omitted. Similarly, a specification for the output format for
 5814 a user with appropriate privileges viewing the queues of other users has been omitted.

5815 The **-f** *file* option from System V is used instead of the BSD method of using the last operand as
 5816 the pathname. The BSD method is ambiguous—does:

5817 `at 1200 friday`

5818 mean the same thing if there is a file named **friday** in the current directory?

5819 The *at_job_id* is composed of a limited character set in historical practice, and it is mandated here
 5820 to invalidate systems that might try using characters that require shell quoting or that could not
 5821 be easily parsed by shell scripts.

5822 The *at* utility varies between System V and BSD systems in the way timezones are used. On
 5823 System V systems, the *TZ* variable affects the at-job submission times and the times displayed
 5824 for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved
 5825 with the current specification. If the user wishes to have the timezone default to that of the
 5826 system, they merely need to issue the *at* command immediately following an unsetting or null
 5827 assignment to *TZ*. For example:

5828 `TZ= at noon ...`

5829 gives the desired BSD result.

5830 While the yacc-like grammar specified in the OPERANDS section is lexically unambiguous with
 5831 respect to the digit strings, a lexical analyzer would probably be written to look for and return
 5832 digit strings in those cases. The parser could then check whether the digit string returned is a
 5833 valid *day_number*, *year_number*, and so on, based on the context.

5834 **FUTURE DIRECTIONS**

5835 None.

5836 **SEE ALSO**5837 *batch, crontab*5838 **CHANGE HISTORY**

5839 First released in Issue 2.

5840 **Issue 6**

5841 This utility is marked as part of the User Portability Utilities option.

5842 The following new requirements on POSIX implementations derive from alignment with the
5843 Single UNIX Specification:

- 5844
- If **-m** is not used, the job's standard output and standard error are provided to the user by
-
- 5845 mail.

5846 The effects of using the **-q** and **-t** options as defined in the IEEE P1003.2b draft standard are
5847 specified.

5848 The normative text is reworded to avoid use of the term “must” for application requirements.

5849 **NAME**

5850 awk — pattern scanning and processing language

5851 **SYNOPSIS**5852 awk [-F *ERE*] [-v *assignment*] ... *program* [*argument* ...]5853 awk [-F *ERE*] -f *progfile* ... [-v *assignment*] ... [*argument* ...]5854 **DESCRIPTION**

5855 The *awk* utility shall execute programs written in the *awk* programming language, which is
 5856 specialized for textual data manipulation. An *awk* program is a sequence of patterns and
 5857 corresponding actions. When input is read that matches a pattern, the action associated with
 5858 that pattern is carried out.

5859 Input shall be interpreted as a sequence of records. By default, a record is a line, less its
 5860 terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of
 5861 input shall be matched in turn against each pattern in the program. For each pattern matched,
 5862 the associated action shall be executed.

5863 The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field
 5864 is a string of non-<blank>s. This default white-space field delimiter can be changed by using the
 5865 **FS** built-in variable or **-F *ERE***. The *awk* utility shall denote the first field in a record \$1, the
 5866 second \$2, and so on. The symbol \$0 shall refer to the entire record; setting any other field causes
 5867 the re-evaluation of \$0. Assigning to \$0 shall reset the values of all other fields and the **NF** built-
 5868 in variable.

5869 **OPTIONS**

5870 The *awk* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section
 5871 12.2, Utility Syntax Guidelines.

5872 The following options shall be supported:

5873 **-F *ERE*** Define the input field separator to be the extended regular expression *ERE*, before
 5874 any input is read; see **Regular Expressions** (on page 161).

5875 **-f *progfile*** Specify the pathname of the file *progfile* containing an *awk* program. If multiple
 5876 instances of this option are specified, the concatenation of the files specified as
 5877 *progfile* in the order specified shall be the *awk* program. The *awk* program can
 5878 alternatively be specified in the command line as a single argument.

5879 **-v *assignment*** The application shall ensure that the *assignment* argument is in the same form as an
 5880 *assignment* operand. The specified variable assignment shall occur prior to
 5881 executing the *awk* program, including the actions associated with **BEGIN** patterns
 5882 (if any). Multiple occurrences of this option can be specified.
 5883

5884 **OPERANDS**

5885 The following operands shall be supported:

5886 *program* If no **-f** option is specified, the first operand to *awk* shall be the text of the *awk*
 5887 program. The application shall supply the *program* operand as a single argument to
 5888 *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.

5889 *argument* Either of the following two types of *argument* can be intermixed:

5890 *file* A pathname of a file that contains the input to be read, which is
 5891 matched against the set of patterns in the program. If no *file* operands
 5892 are specified, or if a *file* operand is '-', the standard input shall be
 5893 used.

5894 *assignment* An operand that begins with an underscore or alphabetic character
 5895 from the portable character set (see the table in the Base Definitions
 5896 volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set),
 5897 followed by a sequence of underscores, digits, and alphabets from
 5898 the portable character set, followed by the '=' character, shall
 5899 specify a variable assignment rather than a pathname. The
 5900 characters before the '=' represent the name of an *awk* variable; if
 5901 that name is an *awk* reserved word (see **Grammar** (on page 170)) the
 5902 behavior is undefined. The characters following the equal sign shall
 5903 be interpreted as if they appeared in the *awk* program preceded and
 5904 followed by a double-quote ('"') character, as a **STRING** token (see
 5905 **Grammar** (on page 170)), except that if the last character is an
 5906 unescaped backslash, it shall be interpreted as a literal backslash
 5907 rather than as the first character of the sequence "\". The variable
 5908 shall be assigned the value of that **STRING** token and, if
 5909 appropriate, shall be considered a *numeric string* (see **Expressions in**
 5910 **awk** (on page 156)), the variable shall also be assigned its numeric
 5911 value. Each such variable assignment shall occur just prior to the
 5912 processing of the following *file*, if any. Thus, an assignment before
 5913 the first *file* argument shall be executed after the **BEGIN** actions (if
 5914 any), while an assignment after the last *file* argument shall occur
 5915 before the **END** actions (if any). If there are no *file* arguments,
 5916 assignments shall be executed before processing the standard input.

5917 **STDIN**

5918 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-';
 5919 see the INPUT FILES section. If the *awk* program contains no actions and no patterns, but is
 5920 otherwise a valid *awk* program, standard input and any *file* operands shall not be read and *awk*
 5921 shall exit with a return status of zero.

5922 **INPUT FILES**

5923 Input files to the *awk* program from any of the following sources shall be text files:

- 5924 • Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV** and
- 5925 **ARGC**
- 5926 • Standard input in the absence of any *file* operands
- 5927 • Arguments to the **getline** function

5928 Whether the variable **RS** is set to a value other than a <newline> or not, for these files,
 5929 implementations shall support records terminated with the specified separator up to
 5930 {LINE_MAX} bytes and may support longer records.

5931 If **-f progfile** is specified, the application shall ensure that the files named by each of the *progfile*
 5932 option-arguments are text files and their concatenation, in the same order as they appear in the
 5933 arguments, is an *awk* program.

5934 **ENVIRONMENT VARIABLES**

5935 The following environment variables shall affect the execution of *awk*:

5936 **LANG** Provide a default value for the internationalization variables that are unset or null.
 5937 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,
 5938 Internationalization Variables for the precedence of internationalization variables
 5939 used to determine the values of locale categories.)

5940	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
5941		
5942	<i>LC_COLLATE</i>	
5943		Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions and in comparisons of string values.
5944		
5945		
5946	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the identification of characters as letters, and the mapping of uppercase and lowercase characters for the toupper and tolower functions.
5947		
5948		
5949		
5950		
5951	<i>LC_MESSAGES</i>	
5952		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
5953		
5954	<i>LC_NUMERIC</i>	
5955		Determine the radix character used when interpreting numeric input, performing conversions between numeric and string values, and formatting numeric output.
5956		Regardless of locale, the period character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing <i>awk</i> programs (including assignments in command line arguments).
5957		
5958		
5959		
5960	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
5961	<i>PATH</i>	Determine the search path when looking for commands executed by <i>system(expr)</i> , or input and output pipes; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.
5962		
5963		
5964		In addition, all environment variables shall be visible via the <i>awk</i> variable ENVIRON .
5965	ASYNCHRONOUS EVENTS	
5966		Default.
5967	STDOUT	
5968		The nature of the output files depends on the <i>awk</i> program.
5969	STDERR	
5970		The standard error shall be used only for diagnostic messages.
5971	OUTPUT FILES	
5972		The nature of the output files depends on the <i>awk</i> program.
5973	EXTENDED DESCRIPTION	
5974	Overall Program Structure	
5975		An <i>awk</i> program is composed of pairs of the form:
5976		<i>pattern</i> { <i>action</i> }
5977		Either the pattern or the action (including the enclosing brace characters) can be omitted.
5978		A missing pattern shall match any record of input, and a missing action shall be equivalent to:
5979		{ <i>print</i> }
5980		Execution of the <i>awk</i> program shall start by first executing the actions associated with all BEGIN patterns in the order they occur in the program. Then each <i>file</i> operand (or standard input if no
5981		

files were specified) shall be processed in turn by reading data from the file until a record separator is seen (<newline> by default). Before the first reference to a field in the record is evaluated, the record shall be split into fields, according to the rules in **Regular Expressions** (on page 161), using the value of **FS** that was current at the time the record was read. Each pattern in the program then shall be evaluated in the order of occurrence, and the action associated with each pattern that matches the current record executed. The action for a matching pattern shall be executed before evaluating subsequent patterns. Finally, the actions associated with all **END** patterns shall be executed in the order they occur in the program.

Expressions in awk

Expressions describe computations used in *patterns* and *actions*. In the following table, valid expression operations are given in groups from highest precedence first to lowest precedence last, with equal-precedence operators grouped between horizontal lines. In expression evaluation, where the grammar is formally ambiguous, higher precedence operators shall be evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent any expression, while *lvalue* represents any entity that can be assigned to (that is, on the left side of an assignment operator). The precise syntax of expressions is given in **Grammar** (on page 170).

Table 4-1 Expressions in Decreasing Precedence in awk

Syntax	Name	Type of Result	Associativity
(<i>expr</i>)	Grouping	Type of <i>expr</i>	N / A
<i>\$expr</i>	Field reference	String	N / A
++ <i>lvalue</i>	Pre-increment	Numeric	N / A
-- <i>lvalue</i>	Pre-decrement	Numeric	N / A
<i>lvalue</i> ++	Post-increment	Numeric	N / A
<i>lvalue</i> --	Post-decrement	Numeric	N / A
<i>expr</i> ^ <i>expr</i>	Exponentiation	Numeric	Right
! <i>expr</i>	Logical not	Numeric	N / A
+ <i>expr</i>	Unary plus	Numeric	N / A
- <i>expr</i>	Unary minus	Numeric	N / A
<i>expr</i> * <i>expr</i>	Multiplication	Numeric	Left
<i>expr</i> / <i>expr</i>	Division	Numeric	Left
<i>expr</i> % <i>expr</i>	Modulus	Numeric	Left
<i>expr</i> + <i>expr</i>	Addition	Numeric	Left
<i>expr</i> - <i>expr</i>	Subtraction	Numeric	Left
<i>expr</i> <i>expr</i>	String concatenation	String	Left
<i>expr</i> < <i>expr</i>	Less than	Numeric	None
<i>expr</i> <= <i>expr</i>	Less than or equal to	Numeric	None
<i>expr</i> != <i>expr</i>	Not equal to	Numeric	None
<i>expr</i> == <i>expr</i>	Equal to	Numeric	None
<i>expr</i> > <i>expr</i>	Greater than	Numeric	None
<i>expr</i> >= <i>expr</i>	Greater than or equal to	Numeric	None

Syntax	Name	Type of Result	Associativity
<i>expr</i> ~ <i>expr</i>	ERE match	Numeric	None
<i>expr</i> !~ <i>expr</i>	ERE non-match	Numeric	None
<i>expr</i> in array	Array membership	Numeric	Left
(<i>index</i>) in array	Multi-dimension array membership	Numeric	Left
<i>expr</i> && <i>expr</i>	Logical AND	Numeric	Left
<i>expr</i> <i>expr</i>	Logical OR	Numeric	Left
<i>expr1</i> ? <i>expr2</i> : <i>expr3</i>	Conditional expression	Type of selected <i>expr2</i> or <i>expr3</i>	Right
<i>lvalue</i> ^= <i>expr</i>	Exponentiation assignment	Numeric	Right
<i>lvalue</i> %= <i>expr</i>	Modulus assignment	Numeric	Right
<i>lvalue</i> *= <i>expr</i>	Multiplication assignment	Numeric	Right
<i>lvalue</i> /= <i>expr</i>	Division assignment	Numeric	Right
<i>lvalue</i> += <i>expr</i>	Addition assignment	Numeric	Right
<i>lvalue</i> -= <i>expr</i>	Subtraction assignment	Numeric	Right
<i>lvalue</i> = <i>expr</i>	Assignment	Type of <i>expr</i>	Right

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value by the equivalent of the following calls to functions defined by the ISO C standard:

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

A numeric value that is exactly equal to the value of an integer (see Section 1.7.2 (on page 7)) shall be converted to a string by the equivalent of a call to the **sprintf** function (see **String Functions** (on page 167)) with the string "%d" as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. Any other numeric value shall be converted to a string by the equivalent of a call to the **sprintf** function with the value of the variable **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a floating-point format specification. This volume of IEEE Std 1003.1-2001 specifies no explicit conversions between numbers and strings. An application can force an expression to be treated as a number by adding zero to it, or can force it to be treated as a string by concatenating the null string (" ") to it.

A string value shall be considered a *numeric string* if it comes from one of the following:

1. Field variables
2. Input from the *getline()* function
3. **FILENAME**
4. **ARGV** array elements
5. **ENVIRON** array elements
6. Array elements created by the *split()* function
7. A command line variable assignment

8. Variable assignment from another numeric string variable

and after all the following conversions have been applied, the resulting string would lexically be recognized as a **NUMBER** token as described by the lexical conventions in **Grammar** (on page 170):

- All leading and trailing <blank>s are discarded.
- If the first non-<blank> is ' + ' or ' - ', it is discarded.
- Changing each occurrence of the decimal point character from the current locale to a period.

If a ' − ' character is ignored in the preceding description, the numeric value of the *numeric string* shall be the negation of the numeric value of the recognized **NUMBER** token. Otherwise, the numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER** token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that term is used in this section.

When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall be treated as false and any other value shall be treated as true. Otherwise, a string value of the null string shall be treated as false and any other value shall be treated as true. A Boolean context shall be one of the following:

- The first subexpression of a conditional expression
- An expression operated on by logical NOT, logical AND, or logical OR
- The second expression of a **for** statement
- The expression of an **if** statement
- The expression of the **while** clause in either a **while** or **do...while** statement
- An expression used as a pattern (as in Overall Program Structure)

All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C standard (see Section 1.7.2 (on page 7)).

The value of the expression:

```
expr1 ^ expr2
```

shall be equivalent to the value returned by the ISO C standard function call:

```
pow(expr1, expr2)
```

The expression:

```
lvalue ^= expr
```

shall be equivalent to the ISO C standard expression:

```
lvalue = pow(lvalue, expr)
```

except that lvalue shall be evaluated only once. The value of the expression:

```
expr1 % expr2
```

shall be equivalent to the value returned by the ISO C standard function call:

```
fmod(expr1, expr2)
```

The expression:

```
lvalue %= expr
```

6105 shall be equivalent to the ISO C standard expression:

6106 `lvalue = fmod(lvalue, expr)`

6107 except that *lvalue* shall be evaluated only once.

6108 Variables and fields shall be set by the assignment statement:

6109 `lvalue = expression`

6110 and the type of *expression* shall determine the resulting variable type. The assignment includes
 6111 the arithmetic assignments ("*+=*", "*-=*", "**=*", "*/=*", "*%=*", "*^=*", "*++*", "*--*") all of which
 6112 shall produce a numeric result. The left-hand side of an assignment and the target of increment
 6113 and decrement operators can be one of a variable, an array with index, or a field selector.

6114 The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not
 6115 be declared. They shall initially be empty, and their sizes shall change dynamically. The
 6116 subscripts, or element identifiers, are strings, providing a type of associative array capability. An
 6117 array name followed by a subscript within square brackets can be used as an *lvalue* and thus as
 6118 an expression, as described in the grammar; see **Grammar** (on page 170). Unsubscripted array
 6119 names can be used in only the following contexts:

- 6120 • A parameter in a function definition or function call
- 6121 • The **NAME** token following any use of the keyword **in** as specified in the grammar (see
- 6122 **Grammar** (on page 170)); if the name used in this context is not an array name, the behavior
- 6123 is undefined

6124 A valid array *index* shall consist of one or more comma-separated expressions, similar to the way
 6125 in which multi-dimensional arrays are indexed in some programming languages. Because *awk*
 6126 arrays are really one-dimensional, such a comma-separated list shall be converted to a single
 6127 string by concatenating the string values of the separate expressions, each separated from the
 6128 other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be
 6129 equivalent:

6130 `var[expr1, expr2, ... exprn]`

6131 `var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]`

6132 The application shall ensure that a multi-dimensioned *index* used with the **in** operator is
 6133 parenthesized. The **in** operator, which tests for the existence of a particular array element, shall
 6134 not cause that element to exist. Any other reference to a nonexistent array element shall
 6135 automatically create it.

6136 Comparisons (with the '*<*', "*<=*", "*!=*", "*==*", '*>*', and "*>=*" operators) shall be made
 6137 numerically if both operands are numeric, if one is numeric and the other has a string value that
 6138 is a numeric string, or if one is numeric and the other has the uninitialized value. Otherwise,
 6139 operands shall be converted to strings as required and a string comparison shall be made using
 6140 the locale-specific collation sequence. The value of the comparison expression shall be 1 if the
 6141 relation is true, or 0 if the relation is false.

Variables and Special Variables

Variables can be used in an *awk* program by referencing them. With the exception of function parameters (see **User-Defined Functions** (on page 169)), they are not explicitly declared. Function parameter names shall be local to the function; all other variable names shall be global. The same name shall not be used as both a function parameter name and as the name of a function or a special *awk* variable. The same name shall not be used both as a variable name with global scope and as the name of a function. The same name shall not be used within the same scope both as a scalar variable and as an array. Uninitialized variables, including scalar variables, array elements, and field variables, shall have an uninitialized value. An uninitialized value shall have both a numeric value of zero and a string value of the empty string. Evaluation of variables with an uninitialized value, to either string or numeric, shall be determined by the context in which they are used.

Field variables shall be designated by a ' \$ ' followed by a number or numerical expression. The effect of the field number *expression* evaluating to anything other than a non-negative integer is unspecified; uninitialized variables or string values need not be converted to numeric values in this context. New field variables can be created by assigning a value to them. References to nonexistent fields (that is, fields after **\$NF**), shall evaluate to the uninitialized value. Such references shall not create new fields. However, assigning to a nonexistent field (for example, **\$(NF+2)=5**) shall increase the value of **NF**; create any intervening fields with the uninitialized value; and cause the value of **\$0** to be recomputed, with the fields being separated by the value of **OFS**. Each field variable shall have a string value or an uninitialized value when created. Field variables shall have the uninitialized value when created from **\$0** using **FS** and the variable does not contain any characters. If appropriate, the field variable shall be considered a numeric string (see **Expressions in awk** (on page 156)).

Implementations shall support the following other special variables that are set by *awk*:

- ARGC** The number of elements in the **ARGV** array.
- ARGV** An array of command line arguments, excluding options and the *program* argument, numbered from zero to **ARGC**-1.

The arguments in **ARGV** can be modified or added to; **ARGC** can be altered. As each input file ends, *awk* shall treat the next non-null element of **ARGV**, up to the current value of **ARGC**-1, inclusive, as the name of the next input file. Thus, setting an element of **ARGV** to null means that it shall not be treated as an input file. The name '-' indicates the standard input. If an argument matches the format of an *assignment* operand, this argument shall be treated as an *assignment* rather than a *file* argument.
- CONVFMT** The **printf** format for converting numbers to strings (except for output statements, where **OFMT** is used); "% . 6g" by default.
- ENVIRON** An array representing the value of the environment, as described in the *exec* functions defined in the System Interfaces volume of IEEE Std 1003.1-2001. The indices of the array shall be strings consisting of the names of the environment variables, and the value of each array element shall be a string consisting of the value of that variable. If appropriate, the environment variable shall be considered a *numeric string* (see **Expressions in awk** (on page 156)); the array element shall also have its numeric value.

In all cases where the behavior of *awk* is affected by environment variables (including the environment of any commands that *awk* executes via the **system** function or via pipeline redirections with the **print** statement, the **printf** statement, or the **getline** function), the environment used shall be the environment at the time

6190		<i>awk</i> began executing; it is implementation-defined whether any modification of
6191		ENVIRON affects this environment.
6192	FILENAME	A pathname of the current input file. Inside a BEGIN action the value is
6193		undefined. Inside an END action the value shall be the name of the last input file
6194		processed.
6195	FNR	The ordinal number of the current record in the current file. Inside a BEGIN action
6196		the value shall be zero. Inside an END action the value shall be the number of the
6197		last record processed in the last file processed.
6198	FS	Input field separator regular expression; a <space> by default.
6199	NF	The number of fields in the current record. Inside a BEGIN action, the use of NF is
6200		undefined unless a getline function without a <i>var</i> argument is executed
6201		previously. Inside an END action, NF shall retain the value it had for the last
6202		record read, unless a subsequent, redirected, getline function without a <i>var</i>
6203		argument is performed prior to entering the END action.
6204	NR	The ordinal number of the current record from the start of input. Inside a BEGIN
6205		action the value shall be zero. Inside an END action the value shall be the number
6206		of the last record processed.
6207	OFMT	The printf format for converting numbers to strings in output statements (see
6208		Output Statements (on page 165)); "%.6g" by default. The result of the
6209		conversion is unspecified if the value of OFMT is not a floating-point format
6210		specification.
6211	OFS	The print statement output field separation; <space> by default.
6212	ORS	The print statement output record separator; a <newline> by default.
6213	RLENGTH	The length of the string matched by the match function.
6214	RS	The first character of the string value of RS shall be the input record separator; a
6215		<newline> by default. If RS contains more than one character, the results are
6216		unspecified. If RS is null, then records are separated by sequences consisting of a
6217		<newline> plus one or more blank lines, leading or trailing blank lines shall not
6218		result in empty records at the beginning or end of the input, and a <newline> shall
6219		always be a field separator, no matter what the value of FS is.
6220	RSTART	The starting position of the string matched by the match function, numbering from
6221		1. This shall always be equivalent to the return value of the match function.
6222	SUBSEP	The subscript separator string for multi-dimensional arrays; the default value is
6223		implementation-defined.
6224	Regular Expressions	
6225	The <i>awk</i> utility shall make use of the extended regular expression notation (see the Base	
6226	Definitions volume of IEEE Std 1003.1-2001, Section 9.4, Extended Regular Expressions) except	
6227	that it shall allow the use of C-language conventions for escaping special characters within the	
6228	EREs, as specified in the table in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5,	
6229	File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') and the following	
6230	table; these escape sequences shall be recognized both inside and outside bracket expressions.	
6231	Note that records need not be separated by <newline>s and string constants can contain	
6232	<newline>s, so even the "\n" sequence is valid in <i>awk</i> EREs. Using a slash character within an	
6233	ERE requires the escaping shown in the following table.	

Table 4-2 Escape Sequences in *awk*

Escape Sequence	Description	Meaning
\ "	Backslash quotation-mark	Quotation-mark character
\ /	Backslash slash	Slash character
\ ddd	A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading '\ ' for each byte.
\ c	A backslash character followed by any character not described in this table or in the table in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').	Undefined

A regular expression can be matched against a specific field or string by using one of the two regular expression matching operators, '~' and '!~'. These operators shall interpret their right-hand operand as a regular expression and their left-hand operand as a string. If the regular expression matches the string, the '~' expression shall evaluate to a value of 1, and the '!~' expression shall evaluate to a value of 0. (The regular expression matching operation is as defined by the term *matched* in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.1, Regular Expression Definitions, where a match occurs on any part of the string unless the regular expression is limited with the circumflex or dollar sign special characters.) If the regular expression does not match the string, the '~' expression shall evaluate to a value of 0, and the '!~' expression shall evaluate to a value of 1. If the right-hand operand is any expression other than the lexical token **ERE**, the string value of the expression shall be interpreted as an extended regular expression, including the escape conventions described above. Note that these same escape conventions shall also be applied in determining the value of a string literal (the lexical token **STRING**), and thus shall be applied a second time when a string literal is used in this context.

When an **ERE** token appears as an expression in any context other than as the right-hand of the '~' or '!~' operator or as one of the built-in function arguments described below, the value of the resulting expression shall be the equivalent of:

```
$0 ~ /ere/
```

The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function (see **String Functions** (on page 167)) shall be interpreted as extended regular expressions. These can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner as the right-hand side of the '~' or '!~' operator.

An extended regular expression can be used to separate fields by using the **-F ERE** option or by assigning a string containing the expression to the built-in variable **FS**. The default value of the **FS** variable shall be a single <space>. The following describes **FS** behavior:

1. If **FS** is a null string, the behavior is unspecified.

- 6280 2. If **FS** is a single character:
- 6281 a. If **FS** is <space>, skip leading and trailing <blank>s; fields shall be delimited by sets
- 6282 of one or more <blank>s.
- 6283 b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single
- 6284 occurrence of *c*.
- 6285 3. Otherwise, the string value of **FS** shall be considered to be an extended regular expression.
- 6286 Each occurrence of a sequence matching the extended regular expression shall delimit
- 6287 fields.

6288 Except for the '*~*' and '!~' operators, and in the **gsub**, **match**, **split**, and **sub** built-in functions,

6289 ERE matching shall be based on input records; that is, record separator characters (the first

6290 character of the value of the variable **RS**, <newline> by default) cannot be embedded in the

6291 expression, and no expression shall match the record separator character. If the record separator

6292 is not <newline>, <newline>s embedded in the expression can be matched. For the '*~*' and

6293 "!~" operators, and in those four built-in functions, ERE matching shall be based on text strings;

6294 that is, any character (including <newline> and the record separator) can be embedded in the

6295 pattern, and an appropriate pattern shall match any character. However, in all *awk* ERE

6296 matching, the use of one or more NUL characters in the pattern, input record, or text string

6297 produces undefined results.

6298 **Patterns**

6299 A *pattern* is any valid *expression*, a range specified by two expressions separated by a comma, or

6300 one of the two special patterns **BEGIN** or **END**.

6301 **Special Patterns**

6302 The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern

6303 shall be matched once and its associated action executed before the first record of input is read

6304 (except possibly by use of the **getline** function—see **Input/Output and General Functions** (on

6305 page 168)—in a prior **BEGIN** action) and before command line assignment is done. Each **END**

6306 pattern shall be matched once and its associated action executed after the last record of input has

6307 been read. These two patterns shall have associated actions.

6308 **BEGIN** and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns

6309 shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order

6310 specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern

6311 in a program.

6312 If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action

6313 contains no **getline** function, *awk* shall exit without reading its input when the last statement in

6314 the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern

6315 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the

6316 statements in the **END** actions are executed.

Expression Patterns

An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the result is true, the pattern shall be considered to match, and the associated action (if any) shall be executed. If the result is false, the action shall not be executed.

Pattern Ranges

A pattern range consists of two expressions separated by a comma; in this case, the action shall be performed for all records between a match of the first expression and the following match of the second expression, inclusive. At this point, the pattern range can be repeated starting at input records subsequent to the end of the matched range.

Actions

An action is a sequence of statements as shown in the grammar in **Grammar** (on page 170). Any single statement can be replaced by a statement list enclosed in braces. The application shall ensure that statements in a statement list are separated by <newline>s or semicolons. Statements in a statement list shall be executed sequentially in the order that they appear.

The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero or non-null, the following statement shall be executed; otherwise, if **else** is present, the statement following the **else** shall be executed.

The **if**, **while**, **do...while**, **for**, **break**, and **continue** statements are based on the ISO C standard (see Section 1.7.2 (on page 7)), except that the Boolean expressions shall be treated as described in **Expressions in awk** (on page 156), and except in the case of:

```
for (variable in array)
```

which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue** statement occurs outside of a loop, the behavior is undefined.

The **delete** statement shall remove an individual array element. Thus, the following code deletes an entire array:

```
for (index in array)
    delete array[index]
```

The **next** statement shall cause all further processing of the current input record to be abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or **END** action.

The **exit** statement shall invoke all **END** actions in the order in which they occur in the program source and then terminate the program without reading further input. An **exit** statement inside an **END** action shall terminate the program without further execution of **END** actions. If an expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*, unless subsequent errors are encountered or a subsequent **exit** statement with an expression is executed.

Output Statements

Both **print** and **printf** statements shall write to standard output by default. The output shall be written to the location specified by *output_redirection* if one is supplied, as follows:

```
> expression
>> expression
| expression
```

In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into which to write (for '*>*' or '*>>*') or as a command to be executed (for '*|*'). Using the first two forms, if the file of that name is not currently open, it shall be opened, creating it if necessary and using the first form, truncating the file. The output then shall be appended to the file. As long as the file remains open, subsequent calls in which *expression* evaluates to the same string value shall simply append output to the file. The file remains open until the **close** function (see **Input/Output and General Functions** (on page 168)) is called with an expression that evaluates to the same string value.

The third form shall write output onto a stream piped to the input of a command. The stream shall be created if no stream is currently open with the value of *expression* as its command name. The stream created shall be equivalent to one created by a call to the *popen()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 with the value of *expression* as the *command* argument and a value of *w* as the *mode* argument. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall write output to the existing stream. The stream shall remain open until the **close** function (see **Input/Output and General Functions** (on page 168)) is called with an expression that evaluates to the same string value. At that time, the stream shall be closed as if by a call to the *pclose()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001.

As described in detail by the grammar in **Grammar** (on page 170), these output statements shall take a comma-separated list of *expressions* referred to in the grammar by the non-terminal symbols **expr_list**, **print_expr_list**, or **print_expr_list_opt**. This list is referred to here as the *expression list*, and each member is referred to as an *expression argument*.

The **print** statement shall write the value of each expression argument onto the indicated output stream separated by the current output field separator (see variable **OFS** above), and terminated by the output record separator (see variable **ORS** above). All expression arguments shall be taken as strings, being converted if necessary; this conversion shall be as described in **Expressions in awk** (on page 156), with the exception that the **printf** format in **OFMT** shall be used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole input record (\$0).

The **printf** statement shall produce output based on a notation similar to the File Format Notation used to describe file formats in this volume of IEEE Std 1003.1-2001 (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation). Output shall be produced as specified with the first *expression* argument as the string *format* and subsequent *expression* arguments as the strings *arg1* to *argn*, inclusive, with the following exceptions:

1. The *format* shall be an actual character string rather than a graphical representation. Therefore, it cannot contain empty character positions. The *<space>* in the *format* string, in any context other than a *flag* of a conversion specification, shall be treated as an ordinary character that is copied to the output.
2. If the character set contains a '*Δ*' character and that character appears in the *format* string, it shall be treated as an ordinary character that is copied to the output.

3. The *escape sequences* beginning with a backslash character shall be treated as sequences of ordinary characters that are copied to the output. Note that these same sequences shall be interpreted lexically by *awk* when they appear in literal strings, but they shall not be treated specially by the **printf** statement.
4. A *field width* or *precision* can be specified as the ' * ' character instead of a digit string. In this case the next argument from the expression list shall be fetched and its numeric value taken as the field width or precision.
5. The implementation shall not precede or follow output from the d or u conversion specifier characters with <blank>s not specified by the *format* string.
6. The implementation shall not precede output from the o conversion specifier character with leading zeros not specified by the *format* string.
7. For the c conversion specifier character: if the argument has a numeric value, the character whose encoding is that value shall be output. If the value is zero or is not the encoding of any character in the character set, the behavior is undefined. If the argument does not have a numeric value, the first character of the string value shall be output; if the string does not contain any characters, the behavior is undefined.
8. For each conversion specification that consumes an argument, the next expression argument shall be evaluated. With the exception of the c conversion specifier character, the value shall be converted (according to the rules specified in **Expressions in awk** (on page 156)) to the appropriate type for the conversion specification.
9. If there are insufficient expression arguments to satisfy all the conversion specifications in the *format* string, the behavior is undefined.
10. If any character sequence in the *format* string begins with a ' % ' character, but does not form a valid conversion specification, the behavior is unspecified.

Both **print** and **printf** can output at least {LINE_MAX} bytes.

Functions

The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and general.

Arithmetic Functions

The arithmetic functions, except for **int**, shall be based on the ISO C standard (see Section 1.7.2 (on page 7)). The behavior is undefined in cases where the ISO C standard specifies that an error be returned or that the behavior is undefined. Although the grammar (see **Grammar** (on page 170)) permits built-in functions to appear with no arguments or parentheses, unless the argument or parentheses are indicated as optional in the following list (by displaying them within the " [] " brackets), such use is undefined.

atan2 (<i>y</i> , <i>x</i>)	Return arctangent of <i>y</i> / <i>x</i> in radians in the range $[-\pi, \pi]$.
cos (<i>x</i>)	Return cosine of <i>x</i> , where <i>x</i> is in radians.
sin (<i>x</i>)	Return sine of <i>x</i> , where <i>x</i> is in radians.
exp (<i>x</i>)	Return the exponential function of <i>x</i> .
log (<i>x</i>)	Return the natural logarithm of <i>x</i> .
sqrt (<i>x</i>)	Return the square root of <i>x</i> .

6441 **int**(*x*) Return the argument truncated to an integer. Truncation shall be toward 0 when
 6442 $x > 0$.

6443 **rand**() Return a random number *n*, such that $0 \leq n < 1$.

6444 **srand**([*expr*]) Set the seed value for *rand* to *expr* or use the time of day if *expr* is omitted. The
 6445 previous seed value shall be returned.

6446 **String Functions**

6447 The string functions in the following list shall be supported. Although the grammar (see
 6448 **Grammar** (on page 170)) permits built-in functions to appear with no arguments or parentheses,
 6449 unless the argument or parentheses are indicated as optional in the following list (by displaying
 6450 them within the "[]" brackets), such use is undefined.

6451 **gsub**(*ere*, *repl* [, *in*])
 6452 Behave like **sub** (see below), except that it shall replace all occurrences of the
 6453 regular expression (like the *ed* utility global substitute) in \$0 or in the *in* argument,
 6454 when specified.

6455 **index**(*s*, *t*) Return the position, in characters, numbering from 1, in string *s* where string *t* first
 6456 occurs, or zero if it does not occur at all.

6457 **length**([*s*]) Return the length, in characters, of its argument taken as a string, or of the whole
 6458 record, \$0, if there is no argument.

6459 **match**(*s*, *ere*) Return the position, in characters, numbering from 1, in string *s* where the
 6460 extended regular expression *ere* occurs, or zero if it does not occur at all. RSTART
 6461 shall be set to the starting position (which is the same as the returned value), zero
 6462 if no match is found; RLENGTH shall be set to the length of the matched string, -1
 6463 if no match is found.

6464 **split**(*s*, *a* [, *fs*])
 6465 Split the string *s* into array elements *a*[1], *a*[2], ..., *a*[*n*], and return *n*. All elements
 6466 of the array shall be deleted before the split is performed. The separation shall be
 6467 done with the ERE *fs* or with the field separator **FS** if *fs* is not given. Each array
 6468 element shall have a string value when created and, if appropriate, the array
 6469 element shall be considered a numeric string (see **Expressions in awk** (on page
 6470 156)). The effect of a null string as the value of *fs* is unspecified.

6471 **sprintf**(*fmt*, *expr*, *expr*, ...)
 6472 Format the expressions according to the **printf** format given by *fmt* and return the
 6473 resulting string.

6474 **sub**(*ere*, *repl* [, *in*])
 6475 Substitute the string *repl* in place of the first instance of the extended regular
 6476 expression *ERE* in string *in* and return the number of substitutions. An ampersand
 6477 ('&') appearing in the string *repl* shall be replaced by the string from *in* that
 6478 matches the ERE. An ampersand preceded with a backslash ('\') shall be
 6479 interpreted as the literal ampersand character. An occurrence of two consecutive
 6480 backslashes shall be interpreted as just a single literal backslash character. Any
 6481 other occurrence of a backslash (for example, preceding any other character) shall
 6482 be treated as a literal backslash character. Note that if *repl* is a string literal (the
 6483 lexical token **STRING**; see **Grammar** (on page 170)), the handling of the
 6484 ampersand character occurs after any lexical processing, including any lexical
 6485 backslash escape sequence processing. If *in* is specified and it is not an lvalue (see
 6486 **Expressions in awk** (on page 156)), the behavior is undefined. If *in* is omitted, *awk*

6487		shall use the current record (\$0) in its place.
6488	substr (<i>s</i> , <i>m</i> [, <i>n</i>])	
6489		Return the at most <i>n</i> -character substring of <i>s</i> that begins at position <i>m</i> , numbering
6490		from 1. If <i>n</i> is omitted, or if <i>n</i> specifies more characters than are left in the string,
6491		the length of the substring shall be limited by the length of the string <i>s</i> .
6492	tolower (<i>s</i>)	Return a string based on the string <i>s</i> . Each character in <i>s</i> that is an uppercase letter
6493		specified to have a tolower mapping by the <i>LC_CTYPE</i> category of the current
6494		locale shall be replaced in the returned string by the lowercase letter specified by
6495		the mapping. Other characters in <i>s</i> shall be unchanged in the returned string.
6496	toupper (<i>s</i>)	Return a string based on the string <i>s</i> . Each character in <i>s</i> that is a lowercase letter
6497		specified to have a toupper mapping by the <i>LC_CTYPE</i> category of the current
6498		locale is replaced in the returned string by the uppercase letter specified by the
6499		mapping. Other characters in <i>s</i> are unchanged in the returned string.
6500		All of the preceding functions that take <i>ERE</i> as a parameter expect a pattern or a string valued
6501		expression that is a regular expression as defined in Regular Expressions (on page 161).
6502	Input/Output and General Functions	
6503	The input/output and general functions are:	
6504	close (<i>expression</i>)	
6505		Close the file or pipe opened by a print or printf statement or a call to getline with
6506		the same string-valued <i>expression</i> . The limit on the number of open <i>expression</i>
6507		arguments is implementation-defined. If the close was successful, the function
6508		shall return zero; otherwise, it shall return non-zero.
6509	<i>expression</i> getline [<i>var</i>]	
6510		Read a record of input from a stream piped from the output of a command. The
6511		stream shall be created if no stream is currently open with the value of <i>expression</i> as
6512		its command name. The stream created shall be equivalent to one created by a call
6513		to the <i>popen</i> () function with the value of <i>expression</i> as the <i>command</i> argument and a
6514		value of <i>r</i> as the <i>mode</i> argument. As long as the stream remains open, subsequent
6515		calls in which <i>expression</i> evaluates to the same string value shall read subsequent
6516		records from the stream. The stream shall remain open until the close function is
6517		called with an expression that evaluates to the same string value. At that time, the
6518		stream shall be closed as if by a call to the <i>pclose</i> () function. If <i>var</i> is omitted, \$0
6519		and NF shall be set; otherwise, <i>var</i> shall be set and, if appropriate, it shall be
6520		considered a numeric string (see Expressions in awk (on page 156)).
6521		The getline operator can form ambiguous constructs when there are
6522		unparenthesized operators (including concatenate) to the left of the ' ' (to the
6523		beginning of the expression containing getline). In the context of the '\$'
6524		operator, ' ' shall behave as if it had a lower precedence than '\$'. The result of
6525		evaluating other operators is unspecified, and conforming applications shall
6526		parenthesize properly all such usages.
6527	getline	Set \$0 to the next input record from the current input file. This form of getline shall
6528		set the NF , NR , and FNR variables.
6529	getline <i>var</i>	Set variable <i>var</i> to the next input record from the current input file and, if
6530		appropriate, <i>var</i> shall be considered a numeric string (see Expressions in awk (on
6531		page 156)). This form of getline shall set the FNR and NR variables.

getline [*var*] < *expression*

Read the next record of input from a named file. The *expression* shall be evaluated to produce a string that is used as a pathname. If the file of that name is not currently open, it shall be opened. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall read subsequent records from the file. The file shall remain open until the **close** function is called with an expression that evaluates to the same string value. If *var* is omitted, \$0 and **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered a numeric string (see **Expressions in awk** (on page 156)).

The **getline** operator can form ambiguous constructs when there are unparenthesized binary operators (including concatenate) to the right of the '<' (up to the end of the expression containing the **getline**). The result of evaluating such a construct is unspecified, and conforming applications shall parenthesize properly all such usages.

system(*expression*)

Execute the command given by *expression* in a manner equivalent to the *system()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 and return the exit status of the command.

All forms of **getline** shall return 1 for successful input, zero for end-of-file, and -1 for an error.

Where strings are used as the name of a file or pipeline, the application shall ensure that the strings are textually identical. The terminology “same string value” implies that “equivalent strings”, even those that differ only by <space>s, represent different files.

User-Defined Functions

The *awk* language also provides user-defined functions. Such functions can be defined as:

```
function name([parameter, ...]) { statements }
```

A function can be referred to anywhere in an *awk* program; in particular, its use can precede its definition. The scope of a function is global.

Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is passed as a parameter that the function uses as an array. Function parameters shall be passed by value if scalar and by reference if array name.

The number of parameters in the function definition need not match the number of parameters in the function call. Excess formal parameters can be used as local variables. If fewer arguments are supplied in a function call than are in the function definition, the extra parameters that are used in the function body as scalars shall evaluate to the uninitialized value until they are otherwise initialized, and the extra parameters that are used in the function body as arrays shall be treated as uninitialized arrays where each element evaluates to the uninitialized value until otherwise initialized.

When invoking a function, no white space can be placed between the function name and the opening parenthesis. Function calls can be nested and recursive calls can be made upon functions. Upon return from any nested or recursive function call, the values of all of the calling function's parameters shall be unchanged, except for array parameters passed by reference. The **return** statement can be used to return a value. If a **return** statement appears outside of a function definition, the behavior is undefined.

In the function definition, <newline>s shall be optional before the opening brace and after the closing brace. Function definitions can appear anywhere in the program where a *pattern-action*

6578 pair is allowed.

6579 Grammar

6580 The grammar in this section and the lexical conventions in the following section shall together
 6581 describe the syntax for *awk* programs. The general conventions for this style of grammar are
 6582 described in Section 1.10 (on page 19). A valid program can be represented as the non-terminal
 6583 symbol *program* in the grammar. This formal syntax shall take precedence over the preceding
 6584 text syntax description.

```

6585 %token NAME NUMBER STRING ERE
6586 %token FUNC_NAME /* Name followed by '(' without white space. */

6587 /* Keywords */
6588 %token Begin End
6589 /* 'BEGIN' 'END' */

6590 %token Break Continue Delete Do Else
6591 /* 'break' 'continue' 'delete' 'do' 'else' */

6592 %token Exit For Function If In
6593 /* 'exit' 'for' 'function' 'if' 'in' */

6594 %token Next Print Printf Return While
6595 /* 'next' 'print' 'printf' 'return' 'while' */

6596 /* Reserved function names */
6597 %token BUILTIN_FUNC_NAME
6598 /* One token for the following:
6599 * atan2 cos sin exp log sqrt int rand srand
6600 * gsub index length match split sprintf sub
6601 * substr tolower toupper close system
6602 */
6603 %token GETLINE
6604 /* Syntactically different from other built-ins. */

6605 /* Two-character tokens. */
6606 %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
6607 /* '+=' '-=' '*=' '/=' '%=' '^=' */

6608 %token OR AND NO_MATCH EQ LE GE NE INCR DECR APPEND
6609 /* '|', '&&', '!~', '==', '<=', '>=', '!=', '++', '--', '>>' */

6610 /* One-character tokens. */
6611 %token '{', '}', '(', ')', '[', ']', ',', ';', NEWLINE
6612 %token '+', '-', '*', '%', '^', '!', '>', '<', '|', '?', ':', '~', '$', '='

6613 %start program
6614 %%

6615 program : item_list
6616 | actionless_item_list
6617 ;

6618 item_list : newline_opt
6619 | actionless_item_list item terminator
6620 | item_list item terminator
6621 | item_list action terminator
6622 ;

```

```

6623     actionless_item_list : item_list           pattern terminator
6624                           | actionless_item_list pattern terminator
6625                           ;
6626     item                   : pattern action
6627                           | Function NAME      '(' param_list_opt ')'
6628                             newline_opt action
6629                           | Function FUNC_NAME '(' param_list_opt ')'
6630                             newline_opt action
6631                           ;
6632     param_list_opt        : /* empty */
6633                           | param_list
6634                           ;
6635     param_list            : NAME
6636                           | param_list ',' NAME
6637                           ;
6638     pattern               : Begin
6639                           | End
6640                           | expr
6641                           | expr ',' newline_opt expr
6642                           ;
6643     action                : '{' newline_opt
6644                           | '{' newline_opt terminated_statement_list
6645                           | '{' newline_opt unterminated_statement_list
6646                           ;
6647     terminator            : terminator ';'
6648                           | terminator NEWLINE
6649                           | ';'
6650                           | NEWLINE
6651                           ;
6652     terminated_statement_list : terminated_statement
6653                           | terminated_statement_list terminated_statement
6654                           ;
6655     unterminated_statement_list : unterminated_statement
6656                           | terminated_statement_list unterminated_statement
6657                           ;
6658     terminated_statement  : action newline_opt
6659                           | If '(' expr ')' newline_opt terminated_statement
6660                           | If '(' expr ')' newline_opt terminated_statement
6661                             Else newline_opt terminated_statement
6662                           | While '(' expr ')' newline_opt terminated_statement
6663                           | For '(' simple_statement_opt ';'
6664                             expr_opt ';' simple_statement_opt ')' newline_opt
6665                             terminated_statement
6666                           | For '(' NAME In NAME ')' newline_opt
6667                             terminated_statement
6668                           | ';' newline_opt
6669                           | terminatable_statement NEWLINE newline_opt
6670                           | terminatable_statement ';'      newline_opt

```

```

6671             ;
6672     unterminated_statement : terminatable_statement
6673         | If '(' expr ')' newline_opt unterminated_statement
6674         | If '(' expr ')' newline_opt terminated_statement
6675         | Else newline_opt unterminated_statement
6676         | While '(' expr ')' newline_opt unterminated_statement
6677         | For '(' simple_statement_opt ';'
6678           expr_opt ';' simple_statement_opt ')' newline_opt
6679           unterminated_statement
6680         | For '(' NAME In NAME ')' newline_opt
6681           unterminated_statement
6682             ;
6683     terminatable_statement : simple_statement
6684         | Break
6685         | Continue
6686         | Next
6687         | Exit expr_opt
6688         | Return expr_opt
6689         | Do newline_opt terminated_statement While '(' expr ')'
6690             ;
6691     simple_statement_opt : /* empty */
6692         | simple_statement
6693             ;
6694     simple_statement : Delete NAME '[' expr_list ']'
6695         | expr
6696         | print_statement
6697             ;
6698     print_statement : simple_print_statement
6699         | simple_print_statement output_redirection
6700             ;
6701     simple_print_statement : Print print_expr_list_opt
6702         | Print '(' multiple_expr_list ')'
6703         | Printf print_expr_list
6704         | Printf '(' multiple_expr_list ')'
6705             ;
6706     output_redirection : '>' expr
6707         | APPEND expr
6708         | '|' expr
6709             ;
6710     expr_list_opt : /* empty */
6711         | expr_list
6712             ;
6713     expr_list : expr
6714         | multiple_expr_list
6715             ;
6716     multiple_expr_list : expr ',' newline_opt expr
6717         | multiple_expr_list ',' newline_opt expr

```



```

6718                                     ;
6719     expr_opt                        : /* empty */
6720                                     | expr
6721                                     ;
6722     expr                            : unary_expr
6723                                     | non_unary_expr
6724                                     ;
6725     unary_expr                      : '+' expr
6726                                     | '-' expr
6727                                     | unary_expr '^'      expr
6728                                     | unary_expr '*'      expr
6729                                     | unary_expr '/'      expr
6730                                     | unary_expr '%'      expr
6731                                     | unary_expr '+'      expr
6732                                     | unary_expr '-'      expr
6733                                     | unary_expr          non_unary_expr
6734                                     | unary_expr '<'      expr
6735                                     | unary_expr LE      expr
6736                                     | unary_expr NE      expr
6737                                     | unary_expr EQ      expr
6738                                     | unary_expr '>'      expr
6739                                     | unary_expr GE      expr
6740                                     | unary_expr '~'      expr
6741                                     | unary_expr NO_MATCH expr
6742                                     | unary_expr In NAME
6743                                     | unary_expr AND newline_opt expr
6744                                     | unary_expr OR  newline_opt expr
6745                                     | unary_expr '?' expr ':' expr
6746                                     | unary_input_function
6747                                     ;
6748     non_unary_expr                  : '(' expr ')'
6749                                     | '!' expr
6750                                     | non_unary_expr '^'      expr
6751                                     | non_unary_expr '*'      expr
6752                                     | non_unary_expr '/'      expr
6753                                     | non_unary_expr '%'      expr
6754                                     | non_unary_expr '+'      expr
6755                                     | non_unary_expr '-'      expr
6756                                     | non_unary_expr          non_unary_expr
6757                                     | non_unary_expr '<'      expr
6758                                     | non_unary_expr LE      expr
6759                                     | non_unary_expr NE      expr
6760                                     | non_unary_expr EQ      expr
6761                                     | non_unary_expr '>'      expr
6762                                     | non_unary_expr GE      expr
6763                                     | non_unary_expr '~'      expr
6764                                     | non_unary_expr NO_MATCH expr
6765                                     | non_unary_expr In NAME
6766                                     | '(' multiple_expr_list ')' In NAME
6767                                     | non_unary_expr AND newline_opt expr

```

```

6768         | non_unary_expr OR  newline_opt expr
6769         | non_unary_expr '?' expr ':' expr
6770         | NUMBER
6771         | STRING
6772         | lvalue
6773         | ERE
6774         | lvalue INCR
6775         | lvalue DECR
6776         | INCR lvalue
6777         | DECR lvalue
6778         | lvalue POW_ASSIGN expr
6779         | lvalue MOD_ASSIGN expr
6780         | lvalue MUL_ASSIGN expr
6781         | lvalue DIV_ASSIGN expr
6782         | lvalue ADD_ASSIGN expr
6783         | lvalue SUB_ASSIGN expr
6784         | lvalue '=' expr
6785         | FUNC_NAME '(' expr_list_opt ')'
6786         | /* no white space allowed before '(' */
6787         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
6788         | BUILTIN_FUNC_NAME
6789         | non_unary_input_function
6790         ;

6791     print_expr_list_opt : /* empty */
6792         | print_expr_list
6793         ;

6794     print_expr_list : print_expr
6795         | print_expr_list ',' newline_opt print_expr
6796         ;

6797     print_expr : unary_print_expr
6798         | non_unary_print_expr
6799         ;

6800     unary_print_expr : '+' print_expr
6801         | '-' print_expr
6802         | unary_print_expr '^' print_expr
6803         | unary_print_expr '*' print_expr
6804         | unary_print_expr '/' print_expr
6805         | unary_print_expr '%' print_expr
6806         | unary_print_expr '+' print_expr
6807         | unary_print_expr '-' print_expr
6808         | unary_print_expr non_unary_print_expr
6809         | unary_print_expr '~' print_expr
6810         | unary_print_expr NO_MATCH print_expr
6811         | unary_print_expr In NAME
6812         | unary_print_expr AND newline_opt print_expr
6813         | unary_print_expr OR newline_opt print_expr
6814         | unary_print_expr '?' print_expr ':' print_expr
6815         ;

6816     non_unary_print_expr : '(' expr ')'
6817         | '!' print_expr

```

```

6818         | non_unary_print_expr '^'      print_expr
6819         | non_unary_print_expr '*'      print_expr
6820         | non_unary_print_expr '/'      print_expr
6821         | non_unary_print_expr '%'      print_expr
6822         | non_unary_print_expr '+'      print_expr
6823         | non_unary_print_expr '-'      print_expr
6824         | non_unary_print_expr          non_unary_print_expr
6825         | non_unary_print_expr '~'      print_expr
6826         | non_unary_print_expr NO_MATCH print_expr
6827         | non_unary_print_expr In NAME
6828         | '(' multiple_expr_list ')' In NAME
6829         | non_unary_print_expr AND newline_opt print_expr
6830         | non_unary_print_expr OR  newline_opt print_expr
6831         | non_unary_print_expr '?' print_expr ':' print_expr
6832         | NUMBER
6833         | STRING
6834         | lvalue
6835         | ERE
6836         | lvalue INCR
6837         | lvalue DECR
6838         | INCR lvalue
6839         | DECR lvalue
6840         | lvalue POW_ASSIGN print_expr
6841         | lvalue MOD_ASSIGN print_expr
6842         | lvalue MUL_ASSIGN print_expr
6843         | lvalue DIV_ASSIGN print_expr
6844         | lvalue ADD_ASSIGN print_expr
6845         | lvalue SUB_ASSIGN print_expr
6846         | lvalue '=' print_expr
6847         | FUNC_NAME '(' expr_list_opt ')'
6848         | /* no white space allowed before '(' */
6849         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
6850         | BUILTIN_FUNC_NAME
6851         ;

6852     lvalue      : NAME
6853         | NAME '[' expr_list ']'
6854         | '$' expr
6855         ;

6856     non_unary_input_function : simple_get
6857         | simple_get '<' expr
6858         | non_unary_expr '|' simple_get
6859         ;

6860     unary_input_function : unary_expr '|' simple_get
6861         ;

6862     simple_get      : GETLINE
6863         | GETLINE lvalue
6864         ;

6865     newline_opt     : /* empty */
6866         | newline_opt NEWLINE
6867         ;

```

This grammar has several ambiguities that shall be resolved as follows:

- Operator precedence and associativity shall be as described in Table 4-1 (on page 156).
- In case of ambiguity, an **else** shall be associated with the most immediately preceding **if** that would satisfy the grammar.
- In some contexts, a slash ('/') that is used to surround an ERE could also be the division operator. This shall be resolved in such a way that wherever the division operator could appear, a slash is assumed to be the division operator. (There is no unary division operator.)

One convention that might not be obvious from the formal grammar is where <newline>s are acceptable. There are several obvious placements such as terminating a statement, and a backslash can be used to escape <newline>s between any lexical tokens. In addition, <newline>s without backslashes can follow a comma, an open brace, logical AND operator ("&&"), logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
{ print $1,
    $2 }
```

Lexical Conventions

The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as follows:

1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a given point.
2. A comment shall consist of any characters beginning with the number sign character and terminated by, but excluding the next occurrence of, a <newline>. Comments shall have no effect, except to delimit lexical tokens.
3. The <newline> shall be recognized as the token **NEWLINE**.
4. A backslash character immediately followed by a <newline> shall have no effect.
5. The token **STRING** shall represent a string constant. A string constant shall begin with the character ' '. Within a string constant, a backslash character shall be considered to begin an escape sequence as specified in the table in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation ('\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequences in Table 4-2 (on page 162) shall be recognized. A <newline> shall not occur within a string constant. A string constant shall be terminated by the first unescaped occurrence of the character ' ' after the one that begins the string constant. The value of the string shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting ' ' characters.
6. The token **ERE** represents an extended regular expression constant. An ERE constant shall begin with the slash character. Within an ERE constant, a backslash character shall be considered to begin an escape sequence as specified in the table in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation. In addition, the escape sequences in Table 4-2 (on page 162) shall be recognized. The application shall ensure that a <newline> does not occur within an ERE constant. An ERE constant shall be terminated by the first unescaped occurrence of the slash character after the one that begins the ERE constant. The extended regular expression represented by the ERE constant shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting slash characters.

7. A <blank> shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE** tokens.

8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall be equivalent to either of the tokens **floating-constant** or **integer-constant** as specified by the ISO C standard, with the following exceptions:

- An integer constant cannot begin with 0x or include the hexadecimal digits 'a', 'b', 'c', 'd', 'e', 'f', 'A', 'B', 'C', 'D', 'E', or 'F'.
- The value of an integer constant beginning with 0 shall be taken in decimal rather than octal.
- An integer constant cannot include a suffix ('u', 'U', 'l', or 'L').
- A floating constant cannot include a suffix ('f', 'F', 'l', or 'L').

If the value is too large or too small to be representable (see Section 1.7.2 (on page 7)), the behavior is undefined.

9. A sequence of underscores, digits, and alphabetic characters from the portable character set (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set), beginning with an underscore or alphabetic, shall be considered a word.
10. The following words are keywords that shall be recognized as individual tokens; the name of the token is the same as the keyword:

BEGIN	delete	END	function	in	printf
break	do	exit	getline	next	return
continue	else	for	if	print	while

11. The following words are names of built-in functions and shall be recognized as the token **BUILTIN_FUNC_NAME**:

atan2	gsub	log	split	sub	toupper
close	index	match	sprintf	substr	
cos	int	rand	sqrt	system	
exp	length	sin	srand	tolower	

The above-listed keywords and names of built-in functions are considered reserved words.

12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in function and is not followed immediately (without any delimiters) by the ' (' character.
13. The token **FUNC_NAME** shall consist of a word that is not a keyword or a name of a built-in function, followed immediately (without any delimiters) by the ' (' character. The ' (' character shall not be included as part of the token.
14. The following two-character sequences shall be recognized as the named tokens:

Token Name	Sequence	Token Name	Sequence
ADD_ASSIGN	+=	NO_MATCH	!~
SUB_ASSIGN	-=	EQ	==
MUL_ASSIGN	*=	LE	<=
DIV_ASSIGN	/=	GE	>=
MOD_ASSIGN	%=	NE	!=
POW_ASSIGN	^=	INCR	++
OR	 	DECR	--
AND	&&	APPEND	>>

6956 15. The following single characters shall be recognized as tokens whose names are the
6957 character:

6958 <newline> { } () [] , ; + - * % ^ ! > < | ? : ~ \$ =

6959 There is a lexical ambiguity between the token **ERE** and the tokens **'/'** and **DIV_ASSIGN**.
6960 When an input sequence begins with a slash character in any syntactic context where the token
6961 **'/'** or **DIV_ASSIGN** could appear as the next token in a valid program, the longer of those two
6962 tokens that can be recognized shall be recognized. In any other syntactic context where the token
6963 **ERE** could appear as the next token in a valid program, the token **ERE** shall be recognized.

6964 EXIT STATUS

6965 The following exit values shall be returned:

6966 0 All input files were processed successfully.

6967 >0 An error occurred.

6968 The exit status can be altered within the program by using an **exit** expression.

6969 CONSEQUENCES OF ERRORS

6970 If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a
6971 diagnostic message to standard error and terminate without any further action.

6972 If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk*
6973 program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

6974 APPLICATION USAGE

6975 The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in
6976 the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with
6977 bytes.

6978 Because the concatenation operation is represented by adjacent expressions rather than an
6979 explicit operator, it is often necessary to use parentheses to enforce the proper evaluation
6980 precedence.

6981 EXAMPLES

6982 The *awk* program specified in the command line is most easily specified within single-quotes (for
6983 example, *'program'*) for applications using *sh*, because *awk* programs commonly contain
6984 characters that are special to the shell, including double-quotes. In the cases where an *awk*
6985 program contains single-quote characters, it is usually easiest to specify most of the program as
6986 strings within single-quotes concatenated by the shell with quoted single-quote characters. For
6987 example:

6988 `awk '/'\''/' { print "quote:", $0 }`

6989 prints all lines from the standard input containing a single-quote character, prefixed with *quote:*.

6990 The following are examples of simple *awk* programs:

6991 1. Write to the standard output all input lines for which field 3 is greater than 5:

6992 `$3 > 5`

6993 2. Write every tenth line:

6994 `(NR % 10) == 0`

6995 3. Write any line with a substring matching the regular expression:

6996 `/(G|D)(2[0-9][[:alpha:]]*)/`

```

6997      4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits
6998      and characters. This example uses character classes digit and alpha to match language-
6999      independent digit and alphabetic characters respectively:
7000      /(G|D)([[:digit:][:alpha:]]*)/
7001
7002      5. Write any line in which the second field matches the regular expression and the fourth
7003      field does not:
7004      $2 ~ /xyz/ && $4 !~ /xyz/
7005
7006      6. Write any line in which the second field contains a backslash:
7007      $2 ~ /\\"/>

```

```

7036     }
7037 18. Write the path prefixes contained in the PATH environment variable, one per line:
7038     BEGIN {
7039         n = split (ENVIRON["PATH"], path, ":")
7040         for (i = 1; i <= n; ++i)
7041             print path[i]
7042     }
7043 19. If there is a file named input containing page headers of the form:
7044     Page #
7045     and a file named program that contains:
7046     /Page/    { $2 = n++; }
7047              { print }
7048     then the command line:
7049     awk -f program n=5 input
7050     prints the file input, filling in page numbers starting at 5.

```

7051 RATIONALE

7052 This description is based on the new *awk*, “*nawk*”, (see the referenced *The AWK Programming*
 7053 *Language*), which introduced a number of new features to the historical *awk*:

- 7054 1. New keywords: **delete**, **do**, **function**, **return**
 - 7055 2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**
 - 7056 3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
 - 7057 4. New expression operators: **?**, **:**, **,**, **^**
 - 7058 5. The **FS** variable and the third argument to **split**, now treated as extended regular
 7059 expressions.
 - 7060 6. The operator precedence, changed to more closely match the C language. Two examples
 7061 of code that operate differently are:
- ```

7062 while (n /= 10 > 1) ...
7063 if (!"wk" ~ /bwk/) ...

```

7064 Several features have been added based on newer implementations of *awk*:

- 7065 • Multiple instances of **-f progfile** are permitted.
- 7066 • The new option **-v assignment**.
- 7067 • The new predefined variable **ENVIRON**.
- 7068 • New built-in functions **toupper** and **tolower**.
- 7069 • More formatting capabilities are added to **printf** to match the ISO C standard.

7070 The overall *awk* syntax has always been based on the C language, with a few features from the  
 7071 shell command language and other sources. Because of this, it is not completely compatible with  
 7072 any other language, which has caused confusion for some users. It is not the intent of the  
 7073 standard developers to address such issues. A few relatively minor changes toward making the  
 7074 language more compatible with the ISO C standard were made; most of these changes are based  
 7075 on similar changes in recent implementations, as described above. There remain several C-



language conventions that are not in *awk*. One of the notable ones is the comma operator, which is commonly used to specify multiple expressions in the C language **for** statement. Also, there are various places where *awk* is more restrictive than the C language regarding the type of expression that can be used in a given context. These limitations are due to the different features that the *awk* language does provide.

Regular expressions in *awk* have been extended somewhat from historical implementations to make them a pure superset of extended regular expressions, as defined by IEEE Std 1003.1-2001 (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.4, Extended Regular Expressions). The main extensions are internationalization features and interval expressions. Historical implementations of *awk* have long supported backslash escape sequences as an extension to extended regular expressions, and this extension has been retained despite inconsistency with other utilities. The number of escape sequences recognized in both extended regular expressions and strings has varied (generally increasing with time) among implementations. The set specified by IEEE Std 1003.1-2001 includes most sequences known to be supported by popular implementations and by the ISO C standard. One sequence that is not supported is hexadecimal value escapes beginning with `'\x'`. This would allow values expressed in more than 9 bits to be used within *awk* as in the ISO C standard. However, because this syntax has a non-deterministic length, it does not permit the subsequent character to be a hexadecimal digit. This limitation can be dealt with in the C language by the use of lexical string concatenation. In the *awk* language, concatenation could also be a solution for strings, but not for extended regular expressions (either lexical ERE tokens or strings used dynamically as regular expressions). Because of this limitation, the feature has not been added to IEEE Std 1003.1-2001.

When a string variable is used in a context where an extended regular expression normally appears (where the lexical token ERE is used in the grammar) the string does not contain the literal slashes.

Some versions of *awk* allow the form:

```
func name(args, ...) { statements }
```

This has been deprecated by the authors of the language, who asked that it not be specified.

Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN** action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This behavior has not been documented, and it was not believed that it was necessary to standardize it.

The specification of conversions between string and numeric values is much more detailed than in the documentation of historical implementations or in the referenced *The AWK Programming Language*. Although most of the behavior is designed to be intuitive, the details are necessary to ensure compatible behavior from different implementations. This is especially important in relational expressions since the types of the operands determine whether a string or numeric comparison is performed. From the perspective of an application writer, it is usually sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating a null string) when the type of an expression does not obviously match what is needed. The intent has been to specify historical practice in almost all cases. The one exception is that, in historical implementations, variables and constants maintain both string and numeric values after their original value is converted by any use. This means that referencing a variable or constant can have unexpected side effects. For example, with historical implementations the following program:

```
{
 a = "+2"
 b = 2
```

```

7124 if (NR % 2)
7125 c = a + b
7126 if (a == b)
7127 print "numeric comparison"
7128 else
7129 print "string comparison"
7130 }

```

7131 would perform a numeric comparison (and output numeric comparison) for each odd-  
 7132 numbered line, but perform a string comparison (and output string comparison) for each even-  
 7133 numbered line. IEEE Std 1003.1-2001 ensures that comparisons will be numeric if necessary.  
 7134 With historical implementations, the following program:

```

7135 BEGIN {
7136 OFMT = "%e"
7137 print 3.14
7138 OFMT = "%f"
7139 print 3.14
7140 }

```

7141 would output "3.140000e+00" twice, because in the second **print** statement the constant  
 7142 "3.14" would have a string value from the previous conversion. IEEE Std 1003.1-2001 requires  
 7143 that the output of the second **print** statement be "3.140000". The behavior of historical  
 7144 implementations was seen as too unintuitive and unpredictable.

7145 It was pointed out that with the rules contained in early drafts, the following script would print  
 7146 nothing:

```

7147 BEGIN {
7148 y[1.5] = 1
7149 OFMT = "%e"
7150 print y[1.5]
7151 }

```

7152 Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to  
 7153 affecting output conversions of numbers to strings and **CONVFMT** is used for internal  
 7154 conversions, such as comparisons or array indexing. The default value is the same as that for  
 7155 **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it  
 7156 will receive the historical behavior associated with internal string conversions.

7157 The POSIX *awk* lexical and syntactic conventions are specified more formally than in other  
 7158 sources. Again the intent has been to specify historical practice. One convention that may not be  
 7159 obvious from the formal grammar as in other verbal descriptions is where <newline>s are  
 7160 acceptable. There are several obvious placements such as terminating a statement, and a  
 7161 backslash can be used to escape <newline>s between any lexical tokens. In addition, <newline>s  
 7162 without backslashes can follow a comma, an open brace, a logical AND operator ("&&"), a  
 7163 logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an  
 7164 **if**, **for**, or **while** statement. For example:

```

7165 { print $1,
7166 $2 }

```

7167 The requirement that *awk* add a trailing <newline> to the program argument text is to simplify  
 7168 the grammar, making it match a text file in form. There is no way for an application or test suite  
 7169 to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

IEEE Std 1003.1-2001 requires several changes from historical implementations in order to support internationalization. Probably the most subtle of these is the use of the decimal-point character, defined by the *LC\_NUMERIC* category of the locale, in representations of floating-point numbers. This locale-specific character is used in recognizing numeric input, in converting between strings and numeric values, and in formatting output. However, regardless of locale, the period character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing *awk* programs (including assignments in command line arguments). This is essentially the same convention as the one used in the ISO C standard. The difference is that the C language includes the *setlocale()* function, which permits an application to modify its locale. Because of this capability, a C application begins executing with its locale set to the C locale, and only executes in the environment-specified locale after an explicit call to *setlocale()*. However, adding such an elaborate new feature to the *awk* language was seen as inappropriate for IEEE Std 1003.1-2001. It is possible to execute an *awk* program explicitly in any desired locale by setting the environment in the shell.

The undefined behavior resulting from NULs in extended regular expressions allows future extensions for the GNU *gawk* program to process binary data.

The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic errors) is undefined because it was considered overly limiting on implementations to specify. In most cases such errors can be expected to produce a diagnostic and a non-zero exit status. However, some implementations may choose to extend the language in ways that make use of certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect in some implementations. Also, different implementations might detect a given error during an initial parsing of the program (before reading any input files) while others might detect it when executing the program after reading some input. Implementors should be aware that diagnosing errors as early as possible and producing useful diagnostics can ease debugging of applications, and thus make an implementation more usable.

The unspecified behavior from using multi-character **RS** values is to allow possible future extensions based on extended regular expressions used for record separators. Historical implementations take the first character of the string and ignore the others.

Unspecified behavior when *split(string,array,<null>)* is used is to allow a proposed future extension that would split up a string into an array of individual characters.

In the context of the **getline** function, equally good arguments for different precedences of the **|** and **<** operators can be made. Historical practice has been that:

```
getline < "a" "b"
```

is parsed as:

```
(getline < "a") "b"
```

although many would argue that the intent was that the file **ab** should be read. However:

```
getline < "x" + 1
```

parses as:

```
getline < ("x" + 1)
```

Similar problems occur with the **|** version of **getline**, particularly in combination with **\$**. For example:

```
$"echo hi" | getline
```

(This situation is particularly problematic when used in a **print** statement, where the **|getline** part might be a redirection of the **print**.)

Since in most cases such constructs are not (or at least should not) be used (because they have a natural ambiguity for which there is no conventional parsing), the meaning of these constructs has been made explicitly unspecified. (The effect is that a conforming application that runs into the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual uses of such constructs.

Grammars can be written that would cause an error under these circumstances. Where backwards-compatibility is not a large consideration, implementors may wish to use such grammars.

Some historical implementations have allowed some built-in functions to be called without an argument list, the result being a default argument list chosen in some “reasonable” way. Use of **length** as a synonym for **length(\$0)** is the only one of these forms that is thought to be widely known or widely used; this particular form is documented in various places (for example, most historical *awk* reference pages, although not in the referenced *The AWK Programming Language*) as legitimate practice. With this exception, default argument lists have always been undocumented and vaguely defined, and it is not at all clear how (or if) they should be generalized to user-defined functions. They add no useful functionality and preclude possible future extensions that might need to name functions without calling them. Not standardizing them seems the simplest course. The standard developers considered that **length** merited special treatment, however, since it has been documented in the past and sees possibly substantial use in historical programs. Accordingly, this usage has been made legitimate, but Issue 5 removed the obsolescent marking for XSI-conforming implementations and many otherwise conforming applications depend on this feature.

In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive backslash characters should be used in the string to ensure a single backslash will precede the ampersand when the resultant string is passed to the function. (For example, to specify one literal ampersand in the replacement string, use **gsub(ERE, "\\&")**.)

Historically the only special character in the *repl* argument of **sub** and **gsub** string functions was the ampersand (**&**) character and preceding it with the backslash character was used to turn off its special meaning.

The description in the ISO POSIX-2:1993 standard introduced behavior such that the backslash character was another special character and it was unspecified whether there were any other special characters. This description introduced several portability problems, some of which are described below, and so it has been replaced with the more historical description. Some of the problems include:

- Historically, to create the replacement string, a script could use **gsub(ERE, "\\&")**, but with the ISO POSIX-2:1993 standard wording, it was necessary to use **gsub(ERE, "\\&&")**. Backslash characters are doubled here because all string literals are subject to lexical analysis, which would reduce each pair of backslash characters to a single backslash before being passed to **gsub**.
- Since it was unspecified what the special characters were, for portable scripts to guarantee that characters are printed literally, each character had to be preceded with a backslash. (For example, a portable script had to use **gsub(ERE, "\\h\\i")** to produce a replacement string of "hi".)

The description for comparisons in the ISO POSIX-2:1993 standard did not properly describe historical practice because of the way numeric strings are compared as numbers. The current rules cause the following code:

```

7262 if (0 == "000")
7263 print "strange, but true"
7264 else
7265 print "not true"

```

7266 to do a numeric comparison, causing the **if** to succeed. It should be intuitively obvious that this  
 7267 is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

7268 To fix this problem, the definition of *numeric string* was enhanced to include only those values  
 7269 obtained from specific circumstances (mostly external sources) where it is not possible to  
 7270 determine unambiguously whether the value is intended to be a string or a numeric.

7271 Variables that are assigned to a numeric string shall also be treated as a numeric string. (For  
 7272 example, the notion of a numeric string can be propagated across assignments.) In comparisons,  
 7273 all variables having the uninitialized value are to be treated as a numeric operand evaluating to  
 7274 the numeric value zero.

7275 Uninitialized variables include all types of variables including scalars, array elements, and fields.  
 7276 The definition of an uninitialized value in **Variables and Special Variables** (on page 160) is  
 7277 necessary to describe the value placed on uninitialized variables and on fields that are valid (for  
 7278 example, **< \$NF**) but have no characters in them and to describe how these variables are to be  
 7279 used in comparisons. A valid field, such as **\$1**, that has no characters in it can be obtained from  
 7280 an input line of "\t\t" when **FS='\t'**. Historically, the comparison (**\$1<10**) was done  
 7281 numerically after evaluating **\$1** to the value zero.

7282 The phrase "... also shall have the numeric value of the numeric string" was removed from  
 7283 several sections of the ISO POSIX-2:1993 standard because it specifies an unnecessary  
 7284 implementation detail. It is not necessary for IEEE Std 1003.1-2001 to specify that these objects be  
 7285 assigned two different values. It is only necessary to specify that these objects may evaluate to  
 7286 two different values depending on context.

7287 The description of numeric string processing is based on the behavior of the *atof()* function in  
 7288 the ISO C standard. While it is not a requirement for an implementation to use this function,  
 7289 many historical implementations of *awk* do. In the ISO C standard, floating-point constants use a  
 7290 period as a decimal point character for the language itself, independent of the current locale, but  
 7291 the *atof()* function and the associated *strtod()* function use the decimal point character of the  
 7292 current locale when converting strings to numeric values. Similarly in *awk*, floating-point  
 7293 constants in an *awk* script use a period independent of the locale, but input strings use the  
 7294 decimal point character of the locale.

## 7295 FUTURE DIRECTIONS

7296 None.

## 7297 SEE ALSO

7298 Section 1.10 (on page 19), *grep*, *lex*, *sed*, the System Interfaces volume of IEEE Std 1003.1-2001,  
 7299 *atof()*, *exec*, *popen()*, *setlocale()*, *strtod()*

## 7300 CHANGE HISTORY

7301 First released in Issue 2.

### 7302 Issue 5

7303 The FUTURE DIRECTIONS section is added.

### 7304 Issue 6

7305 The *awk* utility is aligned with the IEEE P1003.2b draft standard.

7306 The normative text is reworded to avoid use of the term "must" for application requirements.

7307 IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence “An occurrence of two  
7308 consecutive backslashes shall be interpreted as just a single literal backslash character.” into the  
7309 description of the **sub** string function.

7310 **NAME**

7311       basename — return non-directory portion of a pathname

7312 **SYNOPSIS**7313       basename *string* [*suffix*]7314 **DESCRIPTION**

7315       The *string* operand shall be treated as a pathname, as defined in the Base Definitions volume of  
 7316       IEEE Std 1003.1-2001, Section 3.266, Pathname. The string *string* shall be converted to the  
 7317       filename corresponding to the last pathname component in *string* and then the suffix string  
 7318       *suffix*, if present, shall be removed. This shall be done by performing actions equivalent to the  
 7319       following steps in order:

- 7320           1. If *string* is a null string, it is unspecified whether the resulting string is ' . ' or a null string.  
 7321           In either case, skip steps 2 through 6.
- 7322           2. If *string* is "//", it is implementation-defined whether steps 3 to 6 are skipped or  
 7323           processed.
- 7324           3. If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In  
 7325           this case, skip steps 4 to 6.
- 7326           4. If there are any trailing slash characters in *string*, they shall be removed.
- 7327           5. If there are any slash characters remaining in *string*, the prefix of *string* up to and including  
 7328           the last slash character in *string* shall be removed.
- 7329           6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is  
 7330           identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed  
 7331           from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an  
 7332           error if *suffix* is not found in *string*.

7333       The resulting string shall be written to standard output.

7334 **OPTIONS**

7335       None.

7336 **OPERANDS**

7337       The following operands shall be supported:

7338       *string*       A string.7339       *suffix*       A string.7340 **STDIN**

7341       Not used.

7342 **INPUT FILES**

7343       None.

7344 **ENVIRONMENT VARIABLES**7345       The following environment variables shall affect the execution of *basename*:

7346       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 7347       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 7348       Internationalization Variables for the precedence of internationalization variables  
 7349       used to determine the values of locale categories.)

7350       *LC\_ALL*      If set to a non-empty string value, override the values of all the other  
 7351       internationalization variables.

7352 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
7353 characters (for example, single-byte as opposed to multi-byte characters in  
7354 arguments).

7355 *LC\_MESSAGES*  
7356 Determine the locale that should be used to affect the format and contents of  
7357 diagnostic messages written to standard error.

7358 xSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

7359 **ASYNCHRONOUS EVENTS**  
7360 Default.

7361 **STDOUT**  
7362 The *basename* utility shall write a line to the standard output in the following format:  
7363 "%s\n", <resulting string>

7364 **STDERR**  
7365 The standard error shall be used only for diagnostic messages.

7366 **OUTPUT FILES**  
7367 None.

7368 **EXTENDED DESCRIPTION**  
7369 None.

7370 **EXIT STATUS**  
7371 The following exit values shall be returned:  
7372 0 Successful completion.  
7373 >0 An error occurred.

7374 **CONSEQUENCES OF ERRORS**  
7375 Default.

7376 **APPLICATION USAGE**  
7377 The definition of *pathname* specifies implementation-defined behavior for pathnames starting  
7378 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the  
7379 beginning of a pathname unless they can ensure that there are more or less than two or are  
7380 prepared to deal with the implementation-defined consequences.

7381 **EXAMPLES**  
7382 If the string *string* is a valid pathname:  
7383 \$(basename "*string*")  
7384 produces a filename that could be used to open the file named by *string* in the directory returned  
7385 by:  
7386 \$(dirname "*string*")  
7387 If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be  
7388 a valid filename. The *basename* utility is not expected to make any judgements about the validity  
7389 of *string* as a pathname; it just follows the specified algorithm to produce a result string.  
7390 The following shell script compiles */usr/src/cmd/cat.c* and moves the output to a file named **cat**  
7391 in the current directory when invoked with the argument */usr/src/cmd/cat* or with the argument  
7392 */usr/src/cmd/cat.c*:



```
7393 c99 $(dirname "$1")/$(basename "$1" .c).c
7394 mv a.out $(basename "$1" .c)
```

**7395 RATIONALE**

7396 The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid  
7397 pathname:

7398 `$(basename "string")`

7399 would be a valid filename for the file in the directory:

7400 `$(dirname "string")`

7401 This would not work for the early proposal versions of these utilities due to the way it specified  
7402 handling of trailing slashes.

7403 Since the definition of *pathname* specifies implementation-defined behavior for pathnames  
7404 starting with two slash characters, this volume of IEEE Std 1003.1-2001 specifies similar  
7405 implementation-defined behavior for the *basename* and *dirname* utilities.

**7406 FUTURE DIRECTIONS**

7407 None.

**7408 SEE ALSO**

7409 Section 2.5 (on page 33), *dirname*

**7410 CHANGE HISTORY**

7411 First released in Issue 2.

**7412 Issue 6**

7413 IEEE PASC Interpretation 1003.2 #164 is applied.

7414 The normative text is reworded to avoid use of the term “must” for application requirements.

7415 **NAME**

7416 batch — schedule commands to be executed in a batch queue

7417 **SYNOPSIS**7418 UP *batch*

7419

7420 **DESCRIPTION**7421 The *batch* utility shall read commands from standard input and schedule them for execution in a  
7422 batch queue. It shall be the equivalent of the command:

7423 at -q b -m now

7424 where queue *b* is a special *at* queue, specifically for batch jobs. Batch jobs shall be submitted to  
7425 the batch queue with no time constraints and shall be run by the system using algorithms, based  
7426 on unspecified factors, that may vary with each invocation of *batch*.7427 XSI Users shall be permitted to use *batch* if their name appears in the file **/usr/lib/cron/at.allow**. If  
7428 that file does not exist, the file **/usr/lib/cron/at.deny** shall be checked to determine whether the  
7429 user shall be denied access to *batch*. If neither file exists, only a process with the appropriate  
7430 privileges shall be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall  
7431 be permitted. The **at.allow** and **at.deny** files shall consist of one user name per line.7432 **OPTIONS**

7433 None.

7434 **OPERANDS**

7435 None.

7436 **STDIN**7437 The standard input shall be a text file consisting of commands acceptable to the shell command  
7438 language described in Chapter 2 (on page 29).7439 **INPUT FILES**7440 XSI The text files **/usr/lib/cron/at.allow** and **/usr/lib/cron/at.deny** shall contain zero or more user  
7441 names, one per line, of users who are, respectively, authorized or denied access to the *at* and  
7442 *batch* utilities.7443 **ENVIRONMENT VARIABLES**7444 The following environment variables shall affect the execution of *batch*:7445 **LANG** Provide a default value for the internationalization variables that are unset or null.  
7446 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
7447 Internationalization Variables for the precedence of internationalization variables  
7448 used to determine the values of locale categories.)7449 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
7450 internationalization variables.7451 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
7452 characters (for example, single-byte as opposed to multi-byte characters in  
7453 arguments and input files).7454 **LC\_MESSAGES**7455 Determine the locale that should be used to affect the format and contents of  
7456 diagnostic messages written to standard error and informative messages written to  
7457 standard output.7458 **LC\_TIME** Determine the format and contents for date and time strings written by *batch*.

|      |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------|-----|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7459 | XSI | <b>NLSPATH</b>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                            |
| 7460 |     | <b>SHELL</b>                  | Determine the name of a command interpreter to be used to invoke the at-job. If the variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for <i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the login shell from the user database; any of the preceding accompanied by a warning diagnostic about which was chosen. |
| 7461 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7462 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7463 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7464 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7465 |     | <b>TZ</b>                     | Determine the timezone. The job shall be submitted for execution at the time specified by <i>timespec</i> or <b>-t</b> <i>time</i> relative to the timezone specified by the <i>TZ</i> variable. If <i>timespec</i> specifies a timezone, it overrides <i>TZ</i> . If <i>timespec</i> does not specify a timezone and <i>TZ</i> is unset or null, an unspecified default timezone shall be used.                 |
| 7466 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7467 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7468 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7469 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7470 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7471 |     |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                         |
| 7472 |     | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7473 |     |                               | When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.                                                                                                                                                                                                                                               |
| 7474 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7475 |     | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7476 |     |                               | The following shall be written to standard error when a job has been successfully submitted:                                                                                                                                                                                                                                                                                                                     |
| 7477 |     |                               | "job %s at %s\n", <i>at_job_id</i> , < <i>date</i> >                                                                                                                                                                                                                                                                                                                                                             |
| 7478 |     |                               | where <i>date</i> shall be equivalent in format to the output of:                                                                                                                                                                                                                                                                                                                                                |
| 7479 |     |                               | date +"%a %b %e %T %Y"                                                                                                                                                                                                                                                                                                                                                                                           |
| 7480 |     |                               | The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the <i>TZ</i> variable).                                                                                                                                                                                                                                                                           |
| 7481 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7482 |     |                               | Neither this, nor warning messages concerning the selection of the command interpreter, are considered a diagnostic that changes the exit status.                                                                                                                                                                                                                                                                |
| 7483 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7484 |     |                               | Diagnostic messages, if any, shall be written to standard error.                                                                                                                                                                                                                                                                                                                                                 |
| 7485 |     | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7486 |     |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 7487 |     | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7488 |     |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 7489 |     | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7490 |     |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                     |
| 7491 |     |                               | 0 Successful completion.                                                                                                                                                                                                                                                                                                                                                                                         |
| 7492 |     |                               | >0 An error occurred.                                                                                                                                                                                                                                                                                                                                                                                            |
| 7493 |     | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 7494 |     |                               | The job shall not be scheduled.                                                                                                                                                                                                                                                                                                                                                                                  |

**APPLICATION USAGE**

It may be useful to redirect standard output within the specified commands.

**EXAMPLES**

1. This sequence can be used at a terminal:

```
batch
sort < file >outfile
EOT
```

2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
batch <<
! diff file1 file2 2>&1 >outfile | mailx mygroup
!
```

**RATIONALE**

Early proposals described *batch* in a manner totally separated from *at*, even though the historical model treated it almost as a synonym for *at* **-qb**. A number of features were added to list and control batch work separately from those in *at*. Upon further reflection, it was decided that the benefit of this did not merit the change to the historical interface.

The **-m** option was included on the equivalent *at* command because it is historical practice to mail results to the submitter, even if all job-produced output is redirected. As explained in the RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling delays), despite some historical systems where *at* **now** would have been considered an error.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*at*

**CHANGE HISTORY**

First released in Issue 2.

**Issue 6**

This utility is marked as part of the User Portability Utilities option.

The NAME is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

7526 **NAME**

7527       bc — arbitrary-precision arithmetic language

7528 **SYNOPSIS**7529       bc [-l] [*file* ...]7530 **DESCRIPTION**

7531       The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files  
 7532       given, then read from the standard input. If the standard input and standard output to *bc* are  
 7533       attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing  
 7534       behavioral constraints described in the following sections.

7535 **OPTIONS**

7536       The *bc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 7537       Utility Syntax Guidelines.

7538       The following option shall be supported:

7539       **-l**           (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the  
 7540       default zero; see the EXTENDED DESCRIPTION section.

7541 **OPERANDS**

7542       The following operand shall be supported:

7543       *file*           A pathname of a text file containing *bc* program statements. After all *files* have  
 7544       been read, *bc* shall read the standard input.

7545 **STDIN**

7546       See the INPUT FILES section.

7547 **INPUT FILES**

7548       Input files shall be text files containing a sequence of comments, statements, and function  
 7549       definitions that shall be executed as they are read.

7550 **ENVIRONMENT VARIABLES**7551       The following environment variables shall affect the execution of *bc*:

7552       **LANG**           Provide a default value for the internationalization variables that are unset or null.  
 7553       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 7554       Internationalization Variables for the precedence of internationalization variables  
 7555       used to determine the values of locale categories.)

7556       **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 7557       internationalization variables.

7558       **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
 7559       characters (for example, single-byte as opposed to multi-byte characters in  
 7560       arguments and input files).

7561       **LC\_MESSAGES**

7562       Determine the locale that should be used to affect the format and contents of  
 7563       diagnostic messages written to standard error.

7564       **NSI**       **NLS\_PATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

7565 **ASYNCHRONOUS EVENTS**

7566       Default.

7567 **STDOUT**

7568       The output of the *bc* utility shall be controlled by the program read, and consist of zero or more  
 7569       lines containing the value of all executed expressions without assignments. The radix and  
 7570       precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the  
 7571       EXTENDED DESCRIPTION section.

7572 **STDERR**

7573       The standard error shall be used only for diagnostic messages.

7574 **OUTPUT FILES**

7575       None.

7576 **EXTENDED DESCRIPTION**7577       **Grammar**

7578       The grammar in this section and the lexical conventions in the following section shall together  
 7579       describe the syntax for *bc* programs. The general conventions for this style of grammar are  
 7580       described in Section 1.10 (on page 19). A valid program can be represented as the non-terminal  
 7581       symbol **program** in the grammar. This formal syntax shall take precedence over the text syntax  
 7582       description.

```

7583 %token EOF NEWLINE STRING LETTER NUMBER
7584 %token MUL_OP
7585 /* '*' , '/' , '%' */
7586 %token ASSIGN_OP
7587 /* '=' , '+=' , '-=' , '*=' , '/=' , '%=' , '^=' */
7588 %token REL_OP
7589 /* '==' , '<=' , '>=' , '!=' , '<' , '>' */
7590 %token INCR_DECR
7591 /* '++' , '--' */
7592 %token Define Break Quit Length
7593 /* 'define' , 'break' , 'quit' , 'length' */
7594 %token Return For If While Sqrt
7595 /* 'return' , 'for' , 'if' , 'while' , 'sqrt' */
7596 %token Scale Ibase Obase Auto
7597 /* 'scale' , 'ibase' , 'obase' , 'auto' */
7598 %start program
7599 %%
7600 program : EOF
7601 | input_item program
7602 ;
7603 input_item : semicolon_list NEWLINE
7604 | function
7605 ;
7606 semicolon_list : /* empty */
7607 | statement
7608 | semicolon_list ';' statement
7609 | semicolon_list ';'

```

```

7610 ;
7611 statement_list : /* empty */
7612 | statement
7613 | statement_list NEWLINE
7614 | statement_list NEWLINE statement
7615 | statement_list ';'
7616 | statement_list ';' statement
7617 ;
7618 statement : expression
7619 | STRING
7620 | Break
7621 | Quit
7622 | Return
7623 | Return '(' return_expression ')'
7624 | For '(' expression ';'
7625 relational_expression ';'
7626 expression ')' statement
7627 | If '(' relational_expression ')' statement
7628 | While '(' relational_expression ')' statement
7629 | '{' statement_list '}'
7630 ;
7631 function : Define LETTER '(' opt_parameter_list ')'
7632 | '{' NEWLINE opt_auto_define_list
7633 | statement_list '}'
7634 ;
7635 opt_parameter_list : /* empty */
7636 | parameter_list
7637 ;
7638 parameter_list : LETTER
7639 | define_list ',' LETTER
7640 ;
7641 opt_auto_define_list : /* empty */
7642 | Auto define_list NEWLINE
7643 | Auto define_list ';'
7644 ;
7645 define_list : LETTER
7646 | LETTER '[' ']'
7647 | define_list ',' LETTER
7648 | define_list ',' LETTER '[' ']'
7649 ;
7650 opt_argument_list : /* empty */
7651 | argument_list
7652 ;
7653 argument_list : expression
7654 | LETTER '[' ']' ',' argument_list
7655 ;

```

```

7656 relational_expression : expression
7657 | expression REL_OP expression
7658 ;
7659 return_expression : /* empty */
7660 | expression
7661 ;
7662 expression : named_expression
7663 | NUMBER
7664 | '(' expression ')'
7665 | LETTER '(' opt_argument_list ')'
7666 | '-' expression
7667 | expression '+' expression
7668 | expression '-' expression
7669 | expression MUL_OP expression
7670 | expression '^' expression
7671 | INCR_DECR named_expression
7672 | named_expression INCR_DECR
7673 | named_expression ASSIGN_OP expression
7674 | Length '(' expression ')'
7675 | Sqrt '(' expression ')'
7676 | Scale '(' expression ')'
7677 ;
7678 named_expression : LETTER
7679 | LETTER '[' expression ']'
7680 | Scale
7681 | Ibase
7682 | Obase
7683 ;

```

#### 7684 Lexical Conventions in bc

7685 The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as  
 7686 follows:

- 7687 1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a  
 7688 given point.
- 7689 2. A comment shall consist of any characters beginning with the two adjacent characters  
 7690 `"/*"` and terminated by the next occurrence of the two adjacent characters `"*/"`.  
 7691 Comments shall have no effect except to delimit lexical tokens.
- 7692 3. The <newline> shall be recognized as the token **NEWLINE**.
- 7693 4. The token **STRING** shall represent a string constant; it shall consist of any characters  
 7694 beginning with the double-quote character (`"`) and terminated by another occurrence of  
 7695 the double-quote character. The value of the string is the sequence of all characters  
 7696 between, but not including, the two double-quote characters. All characters shall be taken  
 7697 literally from the input, and there is no way to specify a string containing a double-quote  
 7698 character. The length of the value of each string shall be limited to {BC\_STRING\_MAX}  
 7699 bytes.
- 7700 5. A <blank> shall have no effect except as an ordinary character if it appears within a  
 7701 **STRING** token, or to delimit a lexical token other than **STRING**.



6. The combination of a backslash character immediately followed by a <newline> shall have no effect other than to delimit lexical tokens with the following exceptions:

- It shall be interpreted as the character sequence "\<newline>" in **STRING** tokens.
- It shall be ignored as part of a multi-line **NUMBER** token.

7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the following grammar:

```

NUMBER : integer
 | '.' integer
 | integer '.'
 | integer '.' integer
 ;

integer : digit
 | integer digit
 ;

digit : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
 | 8 | 9 | A | B | C | D | E | F
 ;

```

8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by the value of the internal register **ibase** (described below). Each of the **digit** characters shall have the value from 0 to 15 in the order listed here, and the period character shall represent the radix point. The behavior is undefined if digits greater than or equal to the value of **ibase** appear in the token. However, note the exception for single-digit values being assigned to **ibase** and **obase** themselves, in **Operations in bc** (on page 198).

9. The following keywords shall be recognized as tokens:

|               |              |               |               |              |
|---------------|--------------|---------------|---------------|--------------|
| <b>auto</b>   | <b>ibase</b> | <b>length</b> | <b>return</b> | <b>while</b> |
| <b>break</b>  | <b>if</b>    | <b>obase</b>  | <b>scale</b>  |              |
| <b>define</b> | <b>for</b>   | <b>quit</b>   | <b>sqrt</b>   |              |

10. Any of the following characters occurring anywhere except within a keyword shall be recognized as the token **LETTER**:

a b c d e f g h i j k l m n o p q r s t u v w x y z

11. The following single-character and two-character sequences shall be recognized as the token **ASSIGN\_OP**:

=   +=   -=   \*=   /=   %=   ^=

12. If an '=' character, as the beginning of a token, is followed by a '-' character with no intervening delimiter, the behavior is undefined.

13. The following single-characters shall be recognized as the token **MUL\_OP**:

\*   /   %

14. The following single-character and two-character sequences shall be recognized as the token **REL\_OP**:

==   <=   >=   !=   <   >

15. The following two-character sequences shall be recognized as the token **INCR\_DECR**:

- 7743            ++    --
- 7744        16. The following single characters shall be recognized as tokens whose names are the
- 7745            character:
- 7746            <newline> ( ) , + - ; [ ] ^ { }
- 7747        17. The token **EOF** is returned when the end of input is reached.

## 7748        **Operations in bc**

7749        There are three kinds of identifiers: ordinary identifiers, array identifiers, and function

7750        identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed

7751        by square brackets (" [ ] "). An array subscript is required except in an argument or auto list.

7752        Arrays are singly dimensioned and can contain up to {BC\_DIM\_MAX} elements. Indexing shall

7753        begin at zero so an array is indexed from 0 to {BC\_DIM\_MAX}-1. Subscripts shall be truncated

7754        to integers. The application shall ensure that function identifiers are followed by parentheses,

7755        possibly enclosing arguments. The three types of identifiers do not conflict.

7756        The following table summarizes the rules for precedence and associativity of all operators.

7757        Operators on the same line shall have the same precedence; rows are in order of decreasing

7758        precedence.

7759                            **Table 4-3** Operators in *bc*

| Operator                  | Associativity |
|---------------------------|---------------|
| ++, --                    | N/A           |
| unary -                   | N/A           |
| ^                         | Right to left |
| *, /, %                   | Left to right |
| +, binary -               | Left to right |
| =, +=, -=, *=, /=, %=, ^= | Right to left |
| ==, <=, >=, !=, <, >      | None          |

7768        Each expression or named expression has a *scale*, which is the number of decimal digits that

7769        shall be maintained as the fractional portion of the expression.

7770        *Named expressions* are places where values are stored. Named expressions shall be valid on the

7771        left side of an assignment. The value of a named expression shall be the value stored in the place

7772        named. Simple identifiers and array elements are named expressions; they have an initial value

7773        of zero and an initial scale of zero.

7774        The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an

7775        expression consisting of the name of one of these registers shall be zero; values assigned to any

7776        of these registers are truncated to integers. The **scale** register shall contain a global value used in

7777        computing the scale of expressions (as described below). The value of the register **scale** is

7778        limited to  $0 \leq \text{scale} \leq \{\text{BC\_SCALE\_MAX}\}$  and shall have a default value of zero. The **ibase** and

7779        **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be

7780        limited to:

$$7781 \quad 2 \leq \text{ibase} \leq 16$$

7782        The value of **obase** shall be limited to:

$$7783 \quad 2 \leq \text{obase} \leq \{\text{BC\_BASE\_MAX}\}$$

7784        When either **ibase** or **obase** is assigned a single **digit** value from the list in **Lexical Conventions**

7785        **in bc** (on page 196), the value shall be assumed in hexadecimal. (For example, **ibase=A** sets to

base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall have initial values of 10.

Internal computations shall be conducted as if in decimal, regardless of the input and output bases, to the specified number of decimal digits. When an exact result is not achieved (for example, **scale**=0; 3.2/1), the result shall be truncated.

For all values of **obase** specified by this volume of IEEE Std 1003.1-2001, *bc* shall output numeric values by performing each of the following steps in order:

1. If the value is less than zero, a hyphen ('-') character shall be output.
2. One of the following is output, depending on the numerical value:
  - If the absolute value of the numerical value is greater than or equal to one, the integer portion of the value shall be output as a series of digits appropriate to **obase** (as described below), most significant digit first. The most significant non-zero digit shall be output next, followed by each successively less significant digit.
  - If the absolute value of the numerical value is less than one but greater than zero and the scale of the numerical value is greater than zero, it is unspecified whether the character 0 is output.
  - If the numerical value is zero, the character 0 shall be output.
3. If the scale of the value is greater than zero and the numeric value is not zero, a period character shall be output, followed by a series of digits appropriate to **obase** (as described below) representing the most significant portion of the fractional part of the value. If *s* represents the scale of the value being output, the number of digits output shall be *s* if **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s* if **obase** is less than 10. For **obase** values other than 10, this should be the number of digits needed to represent a precision of  $10^s$ .

For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

0 1 2 3 4 5 6 7 8 9 A B C D E F

which represent the values zero to 15, inclusive, respectively.

For bases greater than 16, each digit shall be written as a separate multi-digit decimal number. Each digit except the most significant fractional digit shall be preceded by a single <space>. For bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1000, three-digit decimal strings, and so on. For example, the decimal number 1 024 in base 25 would be written as:

Δ01Δ15Δ24

and in base 125, as:

Δ008Δ024

Very large numbers shall be split across lines with 70 characters per line in the POSIX locale; other locales may split at different character boundaries. Lines that are continued shall end with a backslash ('\').

A function call shall consist of a function name followed by parentheses containing a comma-separated list of expressions, which are the function arguments. A whole array passed as an argument shall be specified by the array name followed by empty square brackets. All function arguments shall be passed by value. As a result, changes made to the formal parameters shall have no effect on the actual arguments. If the function terminates by executing a **return**

7830 statement, the value of the function shall be the value of the expression in the parentheses of the  
 7831 **return** statement or shall be zero if no expression is provided or if there is no **return** statement.

7832 The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be  
 7833 truncated in the least significant decimal place. The scale of the result shall be the scale of the  
 7834 expression or the value of **scale**, whichever is larger.

7835 The result of **length**(*expression*) shall be the total number of significant decimal digits in the  
 7836 expression. The scale of the result shall be zero.

7837 The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be  
 7838 zero.

7839 A numeric constant shall be an expression. The scale shall be the number of digits that follow the  
 7840 radix point in the input representing the constant, or zero if no radix point appears.

7841 The sequence ( *expression* ) shall be an expression with the same value and scale as *expression*.  
 7842 The parentheses can be used to alter the normal precedence.

7843 The semantics of the unary and binary operators are as follows:

7844 *-expression*  
 7845 The result shall be the negative of the *expression*. The scale of the result shall be the scale of  
 7846 *expression*.

7847 The unary increment and decrement operators shall not modify the scale of the named  
 7848 expression upon which they operate. The scale of the result shall be the scale of that named  
 7849 expression.

7850 *++named-expression*  
 7851 The named expression shall be incremented by one. The result shall be the value of the  
 7852 named expression after incrementing.

7853 *--named-expression*  
 7854 The named expression shall be decremented by one. The result shall be the value of the  
 7855 named expression after decrementing.

7856 *named-expression++*  
 7857 The named expression shall be incremented by one. The result shall be the value of the  
 7858 named expression before incrementing.

7859 *named-expression--*  
 7860 The named expression shall be decremented by one. The result shall be the value of the  
 7861 named expression before decrementing.

7862 The exponentiation operator, circumflex ( *' ^ '* ), shall bind right to left.

7863 *expression^expression*  
 7864 The result shall be the first *expression* raised to the power of the second *expression*. If the  
 7865 second expression is not an integer, the behavior is undefined. If *a* is the scale of the left  
 7866 expression and *b* is the absolute value of the right expression, the scale of the result shall be:  
 7867 if *b* >= 0 min(*a* \* *b*, max(scale, *a*)) if *b* < 0 scale

7868 The multiplicative operators ( *' \* '*, *' / '*, *' % '* ) shall bind left to right.

7869 *expression\*expression*  
 7870 The result shall be the product of the two expressions. If *a* and *b* are the scales of the two  
 7871 expressions, then the scale of the result shall be:

7872           `min(a+b,max(scale,a,b))`

7873           *expression / expression*  
 7874           The result shall be the quotient of the two expressions. The scale of the result shall be the  
 7875           value of **scale**.

7876           *expression % expression*  
 7877           For expressions *a* and *b*, *a % b* shall be evaluated equivalent to the steps:

7878           1. Compute *a / b* to current scale.  
 7879           2. Use the result to compute:

7880           `a - (a / b) * b`  
 7881           to scale:  
 7882           `max(scale + scale(b), scale(a))`

7883           The scale of the result shall be:  
 7884           `max(scale + scale(b), scale(a))`

7885           When **scale** is zero, the '`%`' operator is the mathematical remainder operator.

7886           The additive operators ('`+`', '`-`') shall bind left to right.

7887           *expression + expression*  
 7888           The result shall be the sum of the two expressions. The scale of the result shall be the  
 7889           maximum of the scales of the expressions.

7890           *expression - expression*  
 7891           The result shall be the difference of the two expressions. The scale of the result shall be the  
 7892           maximum of the scales of the expressions.

7893           The assignment operators ('`=`', '`+=`', '`-=`', '`*=`', '`/=`', '`%=`', '`^=`') shall bind right to left.

7894           *named-expression = expression*  
 7895           This expression shall result in assigning the value of the expression on the right to the  
 7896           named expression on the left. The scale of both the named expression and the result shall be  
 7897           the scale of *expression*.

7898           The compound assignment forms:

7899           *named-expression <operator>= expression*  
 7900           shall be equivalent to:  
 7901           *named-expression = named-expression <operator> expression*  
 7902           except that the *named-expression* shall be evaluated only once.

7903           Unlike all other operators, the relational operators ('`<`', '`>`', '`<=`', '`>=`', '`==`', '`!=`') shall be  
 7904           only valid as the object of an **if**, **while**, or inside a **for** statement.

7905           *expression1 < expression2*  
 7906           The relation shall be true if the value of *expression1* is strictly less than the value of  
 7907           *expression2*.

7908           *expression1 > expression2*  
 7909           The relation shall be true if the value of *expression1* is strictly greater than the value of  
 7910           *expression2*.

7911 *expression1* <= *expression2*  
 7912     The relation shall be true if the value of *expression1* is less than or equal to the value of  
 7913     *expression2*.

7914 *expression1* >= *expression2*  
 7915     The relation shall be true if the value of *expression1* is greater than or equal to the value of  
 7916     *expression2*.

7917 *expression1* = *expression2*  
 7918     The relation shall be true if the values of *expression1* and *expression2* are equal.

7919 *expression1* != *expression2*  
 7920     The relation shall be true if the values of *expression1* and *expression2* are unequal.

7921 There are only two storage classes in **bc**: global and automatic (local). Only identifiers that are  
 7922 local to a function need be declared with the **auto** command. The arguments to a function shall  
 7923 be local to the function. All other identifiers are assumed to be global and available to all  
 7924 functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto  
 7925 shall be allocated on entry to the function and released on returning from the function. They  
 7926 therefore do not retain values between function calls. Auto arrays shall be specified by the array  
 7927 name followed by empty square brackets. On entry to a function, the old values of the names  
 7928 that appear as parameters and as automatic variables shall be pushed onto a stack. Until the  
 7929 function returns, reference to these names shall refer only to the new values.

7930 References to any of these names from other functions that are called from this function also  
 7931 refer to the new value until one of those functions uses the same name for a local variable.

7932 When a statement is an expression, unless the main operator is an assignment, execution of the  
 7933 statement shall write the value of the expression followed by a <newline>.

7934 When a statement is a string, execution of the statement shall write the value of the string.

7935 Statements separated by semicolons or <newline>s shall be executed sequentially. In an  
 7936 interactive invocation of **bc**, each time a <newline> is read that satisfies the grammatical  
 7937 production:

7938 `input_item : semicolon_list NEWLINE`

7939 the sequential list of statements making up the **semicolon\_list** shall be executed immediately  
 7940 and any output produced by that execution shall be written without any delay due to buffering.

7941 In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

7942 The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested;  
 7943 each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the  
 7944 *relation* is false, execution shall resume after *statement*.

7945 A **for** statement (**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

7946 *first-expression*  
 7947 while (*relation*) {  
 7948     *statement*  
 7949     *last-expression*  
 7950 }

7951 The application shall ensure that all three expressions are present.

7952 The **break** statement shall cause termination of a **for** or **while** statement.

7953 The **auto** statement (**auto** *identifier* [*identifier*] ...) shall cause the values of the identifiers to be  
 7954 pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers

7955 shall be specified by following the array name by empty square brackets. The application shall  
 7956 ensure that the **auto** statement is the first statement in a function definition.

7957 A **define** statement:

```
7958 define LETTER (opt_parameter_list) {
7959 opt_auto_define_list
7960 statement_list
7961 }
```

7962 defines a function named **LETTER**. If a function named **LETTER** was previously defined, the  
 7963 **define** statement shall replace the previous definition. The expression:

```
7964 LETTER (opt_argument_list)
```

7965 shall invoke the function named **LETTER**. The behavior is undefined if the number of  
 7966 arguments in the invocation does not match the number of parameters in the definition.  
 7967 Functions shall be defined before they are invoked. A function shall be considered to be defined  
 7968 within its own body, so recursive calls are valid. The values of numeric constants within a  
 7969 function shall be interpreted in the base specified by the value of the **ibase** register when the  
 7970 function is invoked.

7971 The **return** statements (**return** and **return(expression)**) shall cause termination of a function,  
 7972 popping of its auto variables, and specification of the result of the function. The first form shall  
 7973 be equivalent to **return(0)**. The value and scale of the result returned by the function shall be the  
 7974 value and scale of the expression returned.

7975 The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement  
 7976 occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

7977 The following functions shall be defined when the **-l** option is specified:

```
7978 s(expression)
7979 Sine of argument in radians.
```

```
7980 c(expression)
7981 Cosine of argument in radians.
```

```
7982 a(expression)
7983 Arctangent of argument.
```

```
7984 l(expression)
7985 Natural logarithm of argument.
```

```
7986 e(expression)
7987 Exponential function of argument.
```

```
7988 j(expression, expression)
7989 Bessel function of integer order.
```

7990 The scale of the result returned by these functions shall be the value of the **scale** register at the  
 7991 time the function is invoked. The value of the **scale** register after these functions have completed  
 7992 their execution shall be the same value it had upon invocation. The behavior is undefined if any  
 7993 of these functions is invoked with an argument outside the domain of the mathematical  
 7994 function.

## 7995 EXIT STATUS

7996 The following exit values shall be returned:

```
7997 0 All input files were processed successfully.
```

7998            *unspecified*    An error occurred.

## 7999 CONSEQUENCES OF ERRORS

8000            If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.

8002            In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.

## 8005 APPLICATION USAGE

8006            Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.

8007            For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.

8011            The *bc* utility always uses the period (‘.’) character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the period character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*’s input would introduce ambiguities into the language; consider the following example in a locale with a comma as the decimal-point character:

```
8018 define f(a,b) {
8019 ...
8020 }
8021 ...
8022 f(1,2,3)
```

8023            Because of such ambiguities, the period character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the period is also used in output.

## 8026 EXAMPLES

8027            In the shell, the following assigns an approximation of the first ten digits of ‘ $\pi$ ’ to the variable *x*:

```
8029 x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

8030            The following *bc* program prints the same approximation of ‘ $\pi$ ’, with a label, to standard output:

```
8032 scale = 10
8033 "pi equals "
8034 104348 / 33215
```

8035            The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the **-l** option is specified):

```
8037 scale = 20
8038 define e(x) {
8039 auto a, b, c, i, s
8040 a = 1
8041 b = 1
8042 s = 1
```



```

8043 for (i = 1; 1 == 1; i++) {
8044 a = a*x
8045 b = b*i
8046 c = a/b
8047 if (c == 0) {
8048 return(s)
8049 }
8050 s = s+c
8051 }
8052 }

```

8053 The following prints approximate values of the exponential function of the first ten integers:

```

8054 for (i = 1; i <= 10; ++i) {
8055 e(i)
8056 }

```

#### 8057 **RATIONALE**

8058 The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to  
8059 be part of this volume of IEEE Std 1003.1-2001 because *bc* was thought to have a more intuitive  
8060 programmatic interface. Current implementations that implement *bc* using *dc* are expected to be  
8061 compliant.

8062 The exit status for error conditions has been left unspecified for several reasons:

- 8063 • The *bc* utility is used in both interactive and non-interactive situations. Different exit codes  
8064 may be appropriate for the two uses.
- 8065 • It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions, and  
8066 syntax errors are all possibilities.
- 8067 • It is not clear what utility the exit status has.
- 8068 • In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with  
8069 *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc*  
8070 aborted.

8071 The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief  
8072 that *bc file1 file2* is used most often when at least *file1* contains data/function  
8073 declarations/initializations. Having *bc* continue with prerequisite files missing is probably not  
8074 useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check  
8075 all its files for accessibility before opening any of them.

8076 There was considerable debate on the appropriateness of the language accepted by *bc*. Several  
8077 reviewers preferred to see either a pure subset of the C language or some changes to make the  
8078 language more compatible with C. While the *bc* language has some obvious similarities to C, it  
8079 has never claimed to be compatible with any version of C. An interpreter for a subset of C might  
8080 be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility  
8081 is known in historical practice, and it was not within the scope of this volume of  
8082 IEEE Std 1003.1-2001 to define such a language and utility. If and when they are defined, it may  
8083 be appropriate to include them in a future version of IEEE Std 1003.1. This left the following  
8084 alternatives:

- 8085 1. Exclude any calculator language from this volume of IEEE Std 1003.1-2001.

8086 The consensus of the standard developers was that a simple programmatic calculator  
8087 language is very useful for both applications and interactive users. The only arguments for  
8088 excluding any calculator were that it would become obsolete if and when a C-compatible

one emerged, or that the absence would encourage the development of such a C-compatible one. These arguments did not sufficiently address the needs of current application writers.

2. Standardize the historical *dc*, possibly with minor modifications.

The consensus of the standard developers was that *dc* is a fundamentally less usable language and that that would be far too severe a penalty for avoiding the issue of being similar to but incompatible with C.

3. Standardize the historical *bc*, possibly with minor modifications.

This was the approach taken. Most of the proponents of changing the language would not have been satisfied until most or all of the incompatibilities with C were resolved. Since most of the changes considered most desirable would break historical applications and require significant modification to historical implementations, almost no modifications were made. The one significant modification that was made was the replacement of the historical *bc* assignment operators "=", and so on, with the more modern "+=", and so on. The older versions are considered to be fundamentally flawed because of the lexical ambiguity in uses like *a*==1.

In order to permit implementations to deal with backwards-compatibility as they see fit, the behavior of this one ambiguous construct was made undefined. (At least three implementations have been known to support this change already, so the degree of change involved should not be great.)

The '%' operator is the mathematical remainder operator when **scale** is zero. The behavior of this operator for other values of **scale** is from historical implementations of *bc*, and has been maintained for the sake of historical applications despite its non-intuitive nature.

Historical implementations permit setting **ibase** and **obase** to a broader range of values. This includes values less than 2, which were not seen as sufficiently useful to standardize. These implementations do not interpret input properly for values of **ibase** that are greater than 16. This is because numeric constants are recognized syntactically, rather than lexically, as described in this volume of IEEE Std 1003.1-2001. They are built from lexical tokens of single hexadecimal digits and periods. Since <blank>s between tokens are not visible at the syntactic level, it is not possible to recognize the multi-digit "digits" used in the higher bases properly. The ability to recognize input in these bases was not considered useful enough to require modifying these implementations. Note that the recognition of numeric constants at the syntactic level is not a problem with conformance to this volume of IEEE Std 1003.1-2001, as it does not impact the behavior of conforming applications (and correct *bc* programs). Historical implementations also accept input with all of the digits '0'-'9' and 'A'-'F' regardless of the value of **ibase**; since digits with value greater than or equal to **ibase** are not really appropriate, the behavior when they appear is undefined, except for the common case of:

```
ibase=8;
/* Process in octal base. */
...
ibase=A
/* Restore decimal base. */
```

In some historical implementations, if the expression to be written is an uninitialized array element, a leading <space> and/or up to four leading 0 characters may be output before the character zero. This behavior is considered a bug; it is unlikely that any currently conforming application relies on:

8135           echo 'b[3]' | bc

8136           returning 00000 rather than 0.

8137           Exact calculation of the number of fractional digits to output for a given value in a base other  
8138           than 10 can be computationally expensive. Historical implementations use a faster  
8139           approximation, and this is permitted. Note that the requirements apply only to values of **obase**  
8140           that this volume of IEEE Std 1003.1-2001 requires implementations to support (in particular, not  
8141           to 1, 0, or negative bases, if an implementation supports them as an extension).

8142           Historical implementations of *bc* did not allow array parameters to be passed as the last  
8143           parameter to a function. New implementations are encouraged to remove this restriction even  
8144           though it is not required by the grammar.

8145   **FUTURE DIRECTIONS**

8146           None.

8147   **SEE ALSO**

8148           Section 1.10 (on page 19), *awk*

8149   **CHANGE HISTORY**

8150           First released in Issue 4.

8151   **Issue 5**

8152           The FUTURE DIRECTIONS section is added.

8153   **Issue 6**

8154           Updated to align with the IEEE P1003.2b draft standard, which included resolution of several  
8155           interpretations of the ISO POSIX-2: 1993 standard.

8156           The normative text is reworded to avoid use of the term “must” for application requirements.

8157 **NAME**

8158           bg — run jobs in the background

8159 **SYNOPSIS**8160 UP       bg [*job\_id* ...]

8161

8162 **DESCRIPTION**

8163           If job control is enabled (see the description of *set -m*), the *bg* utility shall resume suspended jobs  
 8164           from the current environment (see Section 2.12 (on page 61)) by running them as background  
 8165           jobs. If the job specified by *job\_id* is already a running background job, the *bg* utility shall have no  
 8166           effect and shall exit successfully.

8167           Using *bg* to place a job into the background shall cause its process ID to become “known in the  
 8168           current shell execution environment”, as if it had been started as an asynchronous list; see  
 8169           Section 2.9.3.1 (on page 50).

8170 **OPTIONS**

8171           None.

8172 **OPERANDS**

8173           The following operand shall be supported:

8174           *job\_id*       Specify the job to be resumed as a background job. If no *job\_id* operand is given,  
 8175                           the most recently suspended job shall be used. The format of *job\_id* is described in  
 8176                           the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.203, Job Control Job  
 8177                           ID.

8178 **STDIN**

8179           Not used.

8180 **INPUT FILES**

8181           None.

8182 **ENVIRONMENT VARIABLES**8183           The following environment variables shall affect the execution of *bg*:

8184           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 8185                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 8186                           Internationalization Variables for the precedence of internationalization variables  
 8187                           used to determine the values of locale categories.)

8188           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 8189                           internationalization variables.

8190           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 8191                           characters (for example, single-byte as opposed to multi-byte characters in  
 8192                           arguments).

8193           *LC\_MESSAGES*

8194                           Determine the locale that should be used to affect the format and contents of  
 8195                           diagnostic messages written to standard error.

8196 XSI       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.8197 **ASYNCHRONOUS EVENTS**

8198           Default.

**8199 STDOUT**

8200 The output of *bg* shall consist of a line in the format:

8201 " [%d] %s\n", <job-number>, <command>

8202 where the fields are as follows:

8203 <job-number> A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using  
8204 these utilities, the job can be identified by prefixing the job number with ' % '.

8205 <command> The associated command that was given to the shell.

**8206 STDERR**

8207 The standard error shall be used only for diagnostic messages.

**8208 OUTPUT FILES**

8209 None.

**8210 EXTENDED DESCRIPTION**

8211 None.

**8212 EXIT STATUS**

8213 The following exit values shall be returned:

8214 0 Successful completion.

8215 >0 An error occurred.

**8216 CONSEQUENCES OF ERRORS**

8217 If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the  
8218 background.

**8219 APPLICATION USAGE**

8220 A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see  
8221 the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface. At  
8222 that point, *bg* can put the job into the background. This is most effective when the job is  
8223 expecting no terminal input and its output has been redirected to non-terminal files. A  
8224 background job can be forced to stop when it has terminal output by issuing the command:

8225 `stty tostop`

8226 A background job can be stopped with the command:

8227 `kill -s stop job ID`

8228 The *bg* utility does not work as expected when it is operating in its own utility execution  
8229 environment because that environment has no suspended jobs. In the following examples:

8230 ... | xargs bg  
8231 (bg)

8232 each *bg* operates in a different environment and does not share its parent shell's understanding  
8233 of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.

**8234 EXAMPLES**

8235 None.

**8236 RATIONALE**

8237 The extensions to the shell specified in this volume of IEEE Std 1003.1-2001 have mostly been  
8238 based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs*  
8239 are also based on the KornShell. The standard developers examined the characteristics of the C  
8240 shell versions of these utilities and found that differences exist. Despite widespread use of the C

8241 shell, the KornShell versions were selected for this volume of IEEE Std 1003.1-2001 to maintain a  
8242 degree of uniformity with the rest of the KornShell features selected (such as the very popular  
8243 command line editing features).

8244 The *bg* utility is expected to wrap its output if the output exceeds the number of display  
8245 columns.

8246 **FUTURE DIRECTIONS**

8247 None.

8248 **SEE ALSO**

8249 Section 2.9.3.1 (on page 50), *fg*, *kill*, *jobs*, *wait*

8250 **CHANGE HISTORY**

8251 First released in Issue 4.

8252 **Issue 6**

8253 This utility is marked as part of the User Portability Utilities option.

8254 The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory  
8255 in this issue. This is a FIPS requirement.

8256 **NAME**

8257 c99 — compile standard C programs

8258 **SYNOPSIS**

```
8259 CD c99 [-c] [-D name[=value]] ... [-E] [-g] [-I directory] ... [-L directory]
8260 ... [-o outfile] [-Ooptlevel] [-s] [-U name] ... operand ...
8261
```

8262 **DESCRIPTION**

8263 The *c99* utility is an interface to the standard C compilation system; it shall accept source code  
 8264 conforming to the ISO C standard. The system conceptually consists of a compiler and link  
 8265 editor. The files referenced by *operands* shall be compiled and linked to produce an executable  
 8266 file. (It is unspecified whether the linking occurs entirely within the operation of *c99*; some  
 8267 implementations may produce objects that are not fully resolved until the file is executed.)

8268 If the **-c** option is specified, for all pathname operands of the form *file.c*, the files:

8269 \$(basename *pathname* .c) .o

8270 shall be created as the result of successful compilation. If the **-c** option is not specified, it is  
 8271 unspecified whether such *.o* files are created or deleted for the *file.c* operands.

8272 If there are no options that prevent link editing (such as **-c** or **-E**), and all operands compile and  
 8273 link without error, the resulting executable file shall be written according to the **-o outfile** option  
 8274 (if present) or to the file **a.out**.

8275 The executable file shall be created as specified in Section 1.7.1.4 (on page 4), except that the file  
 8276 permission bits shall be set to:

8277 S\_IRWXO | S\_IRWXG | S\_IRWXU

8278 and the bits specified by the *umask* of the process shall be cleared.

8279 **OPTIONS**

8280 The *c99* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 8281 12.2, Utility Syntax Guidelines, except that:

- 8282 • The **-I library** operands have the format of options, but their position within a list of  
 8283 operands affects the order in which libraries are searched.
- 8284 • The order of specifying the **-I** and **-L** options is significant.
- 8285 • Conforming applications shall specify each option separately; that is, grouping option letters  
 8286 (for example, **-cO**) need not be recognized by all implementations.

8287 The following options shall be supported:

8288 **-c** Suppress the link-edit phase of the compilation, and do not remove any object files  
 8289 that are produced.

8290 **-g** Produce symbolic information in the object or executable files; the nature of this  
 8291 information is unspecified, and may be modified by implementation-defined  
 8292 interactions with other options.

8293 **-s** Produce object or executable files, or both, from which symbolic and other  
 8294 information not required for proper execution using the *exec* family defined in the  
 8295 System Interfaces volume of IEEE Std 1003.1-2001 has been removed (stripped). If  
 8296 both **-g** and **-s** options are present, the action taken is unspecified.

8297 **-o outfile** Use the pathname *outfile*, instead of the default **a.out**, for the executable file  
 8298 produced. If the **-o** option is present with **-c** or **-E**, the result is unspecified.

- 8299       **-D** *name*[=*value*]  
 8300           Define *name* as if by a C-language **#define** directive. If no *=value* is given, a value of  
 8301           1 shall be used. The **-D** option has lower precedence than the **-U** option. That is, if  
 8302           *name* is used in both a **-U** and a **-D** option, *name* shall be undefined regardless of  
 8303           the order of the options. Additional implementation-defined *names* may be  
 8304           provided by the compiler. Implementations shall support at least 2 048 bytes of **-D**  
 8305           definitions and 256 *names*.
- 8306       **-E**           Copy C-language source files to standard output, expanding all preprocessor  
 8307           directives; no compilation shall be performed. If any operand is not a text file, the  
 8308           effects are unspecified.
- 8309       **-I** *directory*   Change the algorithm for searching for headers whose names are not absolute  
 8310           pathnames to look in the directory named by the *directory* pathname before  
 8311           looking in the usual places. Thus, headers whose names are enclosed in double-  
 8312           quotes (" ") shall be searched for first in the directory of the file with the **#include**  
 8313           line, then in directories named in **-I** options, and last in the usual places. For  
 8314           headers whose names are enclosed in angle brackets ("**<**" "**>**"), the header shall be  
 8315           searched for only in directories named in **-I** options and then in the usual places.  
 8316           Directories named in **-I** options shall be searched in the order specified.  
 8317           Implementations shall support at least ten instances of this option in a single c99  
 8318           command invocation.
- 8319       **-L** *directory*   Change the algorithm of searching for the libraries named in the **-l** objects to look  
 8320           in the directory named by the *directory* pathname before looking in the usual  
 8321           places. Directories named in **-L** options shall be searched in the order specified.  
 8322           Implementations shall support at least ten instances of this option in a single c99  
 8323           command invocation. If a directory specified by a **-L** option contains files named  
 8324           **libc.a**, **libm.a**, **libl.a**, or **liby.a**, the results are unspecified.
- 8325       **-O** *optlevel*   Specify the level of code optimization. If the *optlevel* option-argument is the digit  
 8326           '0', all special code optimizations shall be disabled. If it is the digit '1', the  
 8327           nature of the optimization is unspecified. If the **-O** option is omitted, the nature of  
 8328           the system's default optimization is unspecified. It is unspecified whether code  
 8329           generated in the presence of the **-O 0** option is the same as that generated when  
 8330           **-O** is omitted. Other *optlevel* values may be supported.
- 8331       **-U** *name*       Remove any initial definition of *name*.
- 8332       Multiple instances of the **-D**, **-I**, **-U**, and **-L** options can be specified.

## 8333 OPERANDS

- 8334       An *operand* is either in the form of a pathname or the form **-l** *library*. The application shall  
 8335       ensure that at least one operand of the pathname form is specified. The following operands shall  
 8336       be supported:
- 8337       *file.c*       A C-language source file to be compiled and optionally linked. The application  
 8338       shall ensure that the operand is of this form if the **-c** option is used.
- 8339       *file.a*       A library of object files typically produced by the *ar* utility, and passed directly to  
 8340       the link editor. Implementations may recognize implementation-defined suffixes  
 8341       other than **.a** as denoting object file libraries.
- 8342       *file.o*       An object file produced by c99 **-c** and passed directly to the link editor.  
 8343       Implementations may recognize implementation-defined suffixes other than **.o** as  
 8344       denoting object files.



8345           The processing of other files is implementation-defined.

8346           *-l library*   (The letter ell.) Search the library named:

8347                       *liblibrary.a*

8348           A library shall be searched when its name is encountered, so the placement of a *-l*

8349           operand is significant. Several standard libraries can be specified in this manner, as

8350           described in the EXTENDED DESCRIPTION section. Implementations may

8351           recognize implementation-defined suffixes other than *.a* as denoting libraries.

8352   **STDIN**

8353           Not used.

8354   **INPUT FILES**

8355           The input file shall be one of the following: a text file containing a C-language source program,

8356           an object file in the format produced by *c99 -c*, or a library of object files, in the format produced

8357           by archiving zero or more object files, using *ar*. Implementations may supply additional utilities

8358           that produce files in these formats. Additional input file formats are implementation-defined.

8359   **ENVIRONMENT VARIABLES**

8360           The following environment variables shall affect the execution of *c99*:

8361           *LANG*       Provide a default value for the internationalization variables that are unset or null.

8362                       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,

8363                       Internationalization Variables for the precedence of internationalization variables

8364                       used to determine the values of locale categories.)

8365           *LC\_ALL*     If set to a non-empty string value, override the values of all the other

8366                       internationalization variables.

8367           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as

8368                       characters (for example, single-byte as opposed to multi-byte characters in

8369                       arguments and input files).

8370           *LC\_MESSAGES*

8371                       Determine the locale that should be used to affect the format and contents of

8372                       diagnostic messages written to standard error.

8373   XSI       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

8374           *TMPDIR*     Provide a pathname that should override the default directory for temporary files,

8375   XSI           if any. On XSI-conforming systems, provide a pathname that shall override the

8376                       default directory for temporary files, if any.

8377   **ASYNCHRONOUS EVENTS**

8378           Default.

8379   **STDOUT**

8380           If more than one *file* operand ending in *.c* (or possibly other unspecified suffixes) is given, for

8381           each such file:

8382           "*%s:\n*", *<file>*

8383           may be written. These messages, if written, shall precede the processing of each input file; they

8384           shall not be written to the standard output if they are written to the standard error, as described

8385           in the STDERR section.

8386           If the *-E* option is specified, the standard output shall be a text file that represents the results of

8387           the preprocessing stage of the language; it may contain extra information appropriate for

8388           subsequent compilation passes.

8389 **STDERR**

8390 The standard error shall be used only for diagnostic messages. If more than one *file* operand  
 8391 ending in **.c** (or possibly other unspecified suffixes) is given, for each such file:

8392 "%s:\n", <*file*>

8393 may be written to allow identification of the diagnostic and warning messages with the  
 8394 appropriate input file. These messages, if written, shall precede the processing of each input file;  
 8395 they shall not be written to the standard error if they are written to the standard output, as  
 8396 described in the STDOUT section.

8397 This utility may produce warning messages about certain conditions that do not warrant  
 8398 returning an error (non-zero) exit value.

8399 **OUTPUT FILES**

8400 Object files or executable files or both are produced in unspecified formats.

8401 **EXTENDED DESCRIPTION**8402 **Standard Libraries**

8403 The c99 utility shall recognize the following **-l** operands for standard libraries:

8404 **-l c** This operand shall make visible all functions referenced in the System Interfaces  
 8405 volume of IEEE Std 1003.1-2001, with the possible exception of those functions  
 8406 listed as residing in <**aio.h**>, <**arpa/inet.h**>, <**complex.h**>, <**fenv.h**>, <**math.h**>, |  
 8407 <**mqueue.h**>, <**netdb.h**>, <**netinet/in.h**>, <**pthread.h**>, <**sched.h**>,  
 8408 <**semaphore.h**>, <**spawn.h**>, <**sys/socket.h**>, *pthread\_kill()*, and *pthread\_sigmask()*  
 8409 in <**signal.h**>, <**trace.h**>, functions marked as extensions other than as part of the  
 8410 MF or MPR extensions in <**sys/mman.h**>, functions marked as ADV in <**fcntl.h**>,  
 8411 and functions marked as CS, CPT, and TMR in <**time.h**>. This operand shall not  
 8412 be required to be present to cause a search of this library.

8413 **-l l** This operand shall make visible all functions required by the C-language output of  
 8414 *lex* that are not made available through the **-l c** operand.

8415 **-l pthread** This operand shall make visible all functions referenced in <**pthread.h**> and  
 8416 *pthread\_kill()* and *pthread\_sigmask()* referenced in <**signal.h**>. An implementation  
 8417 may search this library in the absence of this operand.

8418 **-l m** This operand shall make visible all functions referenced in <**math.h**>, |  
 8419 <**complex.h**>, and <**fenv.h**>. An implementation may search this library in the  
 8420 absence of this operand.

8421 **-l rt** This operand shall make visible all functions referenced in <**aio.h**>, <**mqueue.h**>,  
 8422 <**sched.h**>, <**semaphore.h**>, and <**spawn.h**>, functions marked as extensions other  
 8423 than as part of the MF or MPR extensions in <**sys/mman.h**>, functions marked as  
 8424 ADV in <**fcntl.h**>, and functions marked as CS, CPT, and TMR in <**time.h**>. An  
 8425 implementation may search this library in the absence of this operand.

8426 **-l trace** This operand shall make visible all functions referenced in <**trace.h**>. An  
 8427 implementation may search this library in the absence of this operand.

8428 **-l xnet** This operand makes visible all functions referenced in <**arpa/inet.h**>, <**netdb.h**>,  
 8429 <**netinet/in.h**>, and <**sys/socket.h**>. An implementation may search this library in  
 8430 the absence of this operand.

8431 **-l y** This operand shall make visible all functions required by the C-language output of  
 8432 *yacc* that are not made available through the **-l c** operand.

In the absence of options that inhibit invocation of the link editor, such as **-c** or **-E**, the **c99** utility shall cause the equivalent of a **-l c** operand to be passed to the link editor as the last **-l** operand, causing it to be searched after all other object files and libraries are loaded.

It is unspecified whether the libraries **libc.a**, **libm.a**, **librt.a**, **libpthread.a**, **libl.a**, **liby.a**, or **libxnet.a** exist as regular files. The implementation may accept as **-l** operands names of objects that do not exist as regular files.

## External Symbols

The C compiler and link editor shall support the significance of external symbols up to a length of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-defined maximum symbol length is unspecified.

The compiler and link editor shall support a minimum of 511 external symbols per source or object file, and a minimum of 4095 external symbols in total. A diagnostic message shall be written to the standard output if the implementation-defined limit is exceeded; other actions are unspecified.

## Programming Environments

All implementations shall support one of the following programming environments as a default. Implementations may support more than one of the following programming environments. Applications can use *sysconf()* or *getconf* to determine which programming environments are supported.

**Table 4-4** Programming Environments: Type Sizes

| Programming Environment<br><i>getconf</i> Name | Bits in<br>int | Bits in<br>long | Bits in<br>pointer | Bits in<br>off_t |
|------------------------------------------------|----------------|-----------------|--------------------|------------------|
| _POSIX_V6_ILP32_OFF32                          | 32             | 32              | 32                 | 32               |
| _POSIX_V6_ILP32_OFFBIG                         | 32             | 32              | 32                 | ≥64              |
| _POSIX_V6_LP64_OFF64                           | 32             | 64              | 64                 | 64               |
| _POSIX_V6_LPBIG_OFFBIG                         | ≥32            | ≥64             | ≥64                | ≥64              |

All implementations shall support one or more environments where the widths of the following types are no greater than the width of type **long**:

**blksize\_t**, **cc\_t**, **mode\_t**, **nfds\_t**, **pid\_t**, **ptrdiff\_t**, **size\_t**, **speed\_t**, **ssize\_t**, **suseconds\_t**, **tcflag\_t**, **useconds\_t**, **wchar\_t**, **wint\_t**

The executable files created when these environments are selected shall be in a proper format for execution by the *exec* family of functions. Each environment may be one of the ones in Table 4-4, or it may be another environment. The names for the environments that meet this requirement shall be output by a *getconf* command using the **\_POSIX\_V6\_WIDTH\_RESTRICTED\_ENVS** argument. If more than one environment meets the requirement, the names of all such environments shall be output on separate lines. Any of these names can then be used in a subsequent *getconf* command to obtain the flags specific to that environment with the following suffixes added as appropriate:

**\_CFLAGS** To get the C compiler flags.

**\_LDFLAGS** To get the linker/loader flags.

**\_LIBS** To get the libraries.

This requirement may be removed in a future version of IEEE Std 1003.1.

When this utility processes a file containing a function called *main()*, it shall be defined with a return type equivalent to **int**. Using return from the initial call to *main()* shall be equivalent (other than with respect to language scope issues) to calling *exit()* with the returned value. Reaching the end of the initial call to *main()* shall be equivalent to calling *exit(0)*. The implementation shall not declare a prototype for this function.

Implementations provide configuration strings for C compiler flags, linker/loader flags, and libraries for each supported environment. When an application needs to use a specific programming environment rather than the implementation default programming environment while compiling, the application shall first verify that the implementation supports the desired environment. If the desired programming environment is supported, the application shall then invoke *c99* with the appropriate C compiler flags as the first options for the compile, the appropriate linker/loader flags after any other options but before any operands, and the appropriate libraries at the end of the operands.

Conforming applications shall not attempt to link together object files compiled for different programming models. Applications shall also be aware that binary data placed in shared memory or in files might not be recognized by applications built for other programming models.

**Table 4-5** Programming Environments: *c99* and *cc* Arguments

| Programming Environment<br><i>getconf</i> Name | Use                                                    | <i>c99</i> and <i>cc</i> Arguments<br><i>getconf</i> Name                                   |
|------------------------------------------------|--------------------------------------------------------|---------------------------------------------------------------------------------------------|
| _POSIX_V6_ILP32_OFF32                          | C Compiler Flags<br>Linker / Loader Flags<br>Libraries | POSIX_V6_ILP32_OFF32_CFLAGS<br>POSIX_V6_ILP32_OFF32_LDFLAGS<br>POSIX_V6_ILP32_OFF32_LIBS    |
| _POSIX_V6_ILP32_OFFBIG                         | C Compiler Flags<br>Linker / Loader Flags<br>Libraries | POSIX_V6_ILP32_OFFBIG_CFLAGS<br>POSIX_V6_ILP32_OFFBIG_LDFLAGS<br>POSIX_V6_ILP32_OFFBIG_LIBS |
| _POSIX_V6_LP64_OFF64                           | C Compiler Flags<br>Linker / Loader Flags<br>Libraries | POSIX_V6_LP64_OFF64_CFLAGS<br>POSIX_V6_LP64_OFF64_LDFLAGS<br>POSIX_V6_LP64_OFF64_LIBS       |
| _POSIX_V6_LPBIG_OFFBIG                         | C Compiler Flags<br>Linker / Loader Flags<br>Libraries | POSIX_V6_LPBIG_OFFBIG_CFLAGS<br>POSIX_V6_LPBIG_OFFBIG_LDFLAGS<br>POSIX_V6_LPBIG_OFFBIG_LIBS |

## EXIT STATUS

The following exit values shall be returned:

0 Successful compilation or link edit.

>0 An error occurred.

## CONSEQUENCES OF ERRORS

When *c99* encounters a compilation error that causes an object file not to be created, it shall write a diagnostic to standard error and continue to compile other source code operands, but it shall not perform the link phase and return a non-zero exit status. If the link edit is unsuccessful, a diagnostic message shall be written to standard error and *c99* exits with a non-zero status. A conforming application shall rely on the exit status of *c99*, rather than on the existence or mode of the executable file.

8517 **APPLICATION USAGE**

8518 Since the **c99** utility usually creates files in the current directory during the compilation process,  
 8519 it is typically necessary to run the **c99** utility in a directory in which a file can be created.

8520 On systems providing POSIX Conformance (see the Base Definitions volume of  
 8521 IEEE Std 1003.1-2001, Chapter 2, Conformance), **c99** is required only with the C-Language  
 8522 Development option; XSI-conformant systems always provide **c99**.

8523 Some historical implementations have created **.o** files when **-c** is not specified and more than  
 8524 one source file is given. Since this area is left unspecified, the application cannot rely on **.o** files  
 8525 being created, but it also must be prepared for any related **.o** files that already exist being deleted  
 8526 at the completion of the link edit.

8527 Some historical implementations have permitted **-L** options to be interspersed with **-l** operands  
 8528 on the command line. For an application to compile consistently on systems that do not behave  
 8529 like this, it is necessary for a conforming application to supply all **-L** options before any of the **-l**  
 8530 options.

8531 There is the possible implication that if a user supplies versions of the standard functions (before  
 8532 they would be encountered by an implicit **-l c** or explicit **-l m**), that those versions would be  
 8533 used in place of the standard versions. There are various reasons this might not be true  
 8534 (functions defined as macros, manipulations for clean name space, and so on), so the existence of  
 8535 files named in the same manner as the standard libraries within the **-L** directories is explicitly  
 8536 stated to produce unspecified behavior.

8537 All of the functions specified in the System Interfaces volume of IEEE Std 1003.1-2001 may be  
 8538 made visible by implementations when the Standard C Library is searched. Conforming  
 8539 applications must explicitly request searching the other standard libraries when functions made  
 8540 visible by those libraries are used.

8541 **EXAMPLES**

8542 1. The following usage example compiles **foo.c** and creates the executable file **foo**:

8543 `c99 -o foo foo.c`

8544 The following usage example compiles **foo.c** and creates the object file **foo.o**:

8545 `c99 -c foo.c`

8546 The following usage example compiles **foo.c** and creates the executable file **a.out**:

8547 `c99 foo.c`

8548 The following usage example compiles **foo.c**, links it with **bar.o**, and creates the executable  
 8549 file **a.out**. It may also create and leave **foo.o**:

8550 `c99 foo.c bar.o`

8551 2. The following example shows how an application using threads interfaces can test for  
 8552 support of and use a programming environment supporting 32-bit **int**, **long**, and **pointer**  
 8553 types and an **off\_t** type using at least 64 bits:

```
8554 if [$(getconf _POSIX_V6_ILP32_OFFBIG) != "-1"]
8555 then
8556 c99 $(getconf POSIX_V6_ILP32_OFFBIG_CFLAGS) -D_XOPEN_SOURCE=600 \
8557 $(getconf POSIX_V6_ILP32_OFFBIG_LDFLAGS) foo.c -o foo \
8558 $(getconf POSIX_V6_ILP32_OFFBIG_LIBS) -l pthread
8559 else
8560 echo ILP32_OFFBIG programming environment not supported
```

```

8561 exit 1
8562 fi

```

3. The following examples clarify the use and interactions of **-L** options and **-l** operands.

Consider the case in which module **a.c** calls function *f()* in library **libQ.a**, and module **b.c** calls function *g()* in library **libp.a**. Assume that both libraries reside in **/a/b/c**. The command line to compile and link in the desired way is:

```

8567 c99 -L /a/b/c main.o a.c -l Q b.c -l p

```

In this case the **-l Q** operand need only precede the first **-l p** operand, since both **libQ.a** and **libp.a** reside in the same directory.

Multiple **-L** operands can be used when library name collisions occur. Building on the previous example, suppose that the user wants to use a new **libp.a**, in **/a/a/a**, but still wants *f()* from **/a/b/c/libQ.a**:

```

8573 c99 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p

```

In this example, the linker searches the **-L** options in the order specified, and finds **/a/a/a/libp.a** before **/a/b/c/libp.a** when resolving references for **b.c**. The order of the **-l** operands is still important, however.

4. The following example shows how an application can use a programming environment where the widths of the following types:

```

8579 blksize_t, cc_t, mode_t, nfd_t, pid_t, ptrdiff_t, size_t, speed_t, ssize_t, suseconds_t,
8580 tcflag_t, useconds_t, wchar_t, wint_t

```

are no greater than the width of type **long**:

```

8582 # First choose one of the listed environments ...
8583 # ... if there are no additional constraints, the first one will do:
8584 CENV=$(getconf _POSIX_V6_WIDTH_RESTRICTED_ENVS | head -n 1)
8585 # ... or, if an environment that supports large files is preferred,
8586 # look for names that contain "OFF64" or "OFFBIG". (This chooses
8587 # the last one in the list if none match.)
8588 for CENV in $(getconf _POSIX_V6_WIDTH_RESTRICTED_ENVS)
8589 do
8590 case $CENV in
8591 *OFF64*|*OFFBIG*) break ;;
8592 esac
8593 done
8594 # The chosen environment name can now be used like this:
8595 c99 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=200112L \
8596 $(getconf ${CENV}_LDFLAGS) foo.c -o foo \
8597 $(getconf ${CENV}_LIBS)

```

## 8598 RATIONALE

8599 The **c99** utility is based on the **c89** utility originally introduced in the ISO POSIX-2: 1993 standard.

8600 Some of the changes from **c89** include the modification to the contents of the Standard Libraries  
8601 section to account for new headers and options; for example, **<spawn.h>** added to the **-l rt**  
8602 operand, and the **-l trace** operand added for the Tracing functions.

8603 **FUTURE DIRECTIONS**

8604       None.

8605 **SEE ALSO**

8606       Section 1.7.1.4 (on page 4), *ar*, *getconf*, *make*, *nm*, *strip*, *umask*, the System Interfaces volume of  
8607       IEEE Std 1003.1-2001, *exec*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
8608       Chapter 13, Headers

8609 **CHANGE HISTORY**

8610       First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8611       IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/12 is applied, correcting the EXTENDED |  
8612       DESCRIPTION of **-l c** and **-l m**. Previously, the text did not take into account the presence of |  
8613       the c99 math headers. |

8614       IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/13 is applied, changing the reference to |  
8615       the **libxnet** library to **libxnet.a**. |

8616 **NAME**

8617 cal — print a calendar

8618 **SYNOPSIS**8619 xSI cal *[[month] year ]*

8620

8621 **DESCRIPTION**

8622 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from  
 8623 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,  
 8624 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on  
 8625 September 14, 1752.

8626 **OPTIONS**

8627 None.

8628 **OPERANDS**

8629 The following operands shall be supported:

8630 *month* Specify the month to be displayed, represented as a decimal integer from 1  
 8631 (January) to 12 (December). The default shall be the current month.

8632 *year* Specify the year for which the calendar is displayed, represented as a decimal  
 8633 integer from 1 to 9999. The default shall be the current year.

8634 **STDIN**

8635 Not used.

8636 **INPUT FILES**

8637 None.

8638 **ENVIRONMENT VARIABLES**8639 The following environment variables shall affect the execution of *cal*:

8640 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 8641 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 8642 Internationalization Variables for the precedence of internationalization variables  
 8643 used to determine the values of locale categories.)

8644 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 8645 internationalization variables.

8646 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 8647 characters (for example, single-byte as opposed to multi-byte characters in  
 8648 arguments).

8649 *LC\_MESSAGES*

8650 Determine the locale that should be used to affect the format and contents of  
 8651 diagnostic messages written to standard error, and informative messages written  
 8652 to standard output.

8653 *LC\_TIME* Determine the format and contents of the calendar.

8654 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

8655 *TZ* Determine the timezone used to calculate the value of the current month.



**8656 ASYNCHRONOUS EVENTS**

8657 Default.

**8658 STDOUT**

8659 The standard output shall be used to display the calendar, in an unspecified format.

**8660 STDERR**

8661 The standard error shall be used only for diagnostic messages.

**8662 OUTPUT FILES**

8663 None.

**8664 EXTENDED DESCRIPTION**

8665 None.

**8666 EXIT STATUS**

8667 The following exit values shall be returned:

8668 0 Successful completion.

8669 >0 An error occurred.

**8670 CONSEQUENCES OF ERRORS**

8671 Default.

**8672 APPLICATION USAGE**

8673 Note that:

8674 cal 83

8675 refers to A.D. 83, not 1983.

**8676 EXAMPLES**

8677 None.

**8678 RATIONALE**

8679 None.

**8680 FUTURE DIRECTIONS**

8681 A future version of IEEE Std 1003.1-2001 may support locale-specific recognition of the date of  
8682 adoption of the Gregorian calendar.

**8683 SEE ALSO**

8684 None.

**8685 CHANGE HISTORY**

8686 First released in Issue 2.

**8687 Issue 6**

8688 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of  
8689 the Gregorian calendar.

8690 **NAME**

8691           cat — concatenate and print files

8692 **SYNOPSIS**8693           cat [-u] [*file* ...]8694 **DESCRIPTION**8695           The *cat* utility shall read files in sequence and shall write their contents to the standard output in  
8696           the same sequence.8697 **OPTIONS**8698           The *cat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
8699           Utility Syntax Guidelines.

8700           The following option shall be supported:

8701           -u           Write bytes from the input file to the standard output without delay as each is  
8702                           read.8703 **OPERANDS**

8704           The following operand shall be supported:

8705           *file*           A pathname of an input file. If no *file* operands are specified, the standard input  
8706                           shall be used. If a *file* is '-', the *cat* utility shall read from the standard input at  
8707                           that point in the sequence. The *cat* utility shall not close and reopen standard input  
8708                           when it is referenced in this way, but shall accept multiple occurrences of '-' as a  
8709                           *file* operand.8710 **STDIN**8711           The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
8712           See the INPUT FILES section.8713 **INPUT FILES**

8714           The input files can be any file type.

8715 **ENVIRONMENT VARIABLES**8716           The following environment variables shall affect the execution of *cat*:8717           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
8718                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
8719                           Internationalization Variables for the precedence of internationalization variables  
8720                           used to determine the values of locale categories.)8721           *LC\_ALL*          If set to a non-empty string value, override the values of all the other  
8722                           internationalization variables.8723           *LC\_CTYPE*       Determine the locale for the interpretation of sequences of bytes of text data as  
8724                           characters (for example, single-byte as opposed to multi-byte characters in  
8725                           arguments).8726           *LC\_MESSAGES*   Determine the locale that should be used to affect the format and contents of  
8727                           diagnostic messages written to standard error.  
87288729 *XSI*           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.8730 **ASYNCHRONOUS EVENTS**

8731           Default.

**8732 STDOUT**

8733       The standard output shall contain the sequence of bytes read from the input files. Nothing else  
8734       shall be written to the standard output.

**8735 STDERR**

8736       The standard error shall be used only for diagnostic messages.

**8737 OUTPUT FILES**

8738       None.

**8739 EXTENDED DESCRIPTION**

8740       None.

**8741 EXIT STATUS**

8742       The following exit values shall be returned:

8743       0   All input files were output successfully.

8744       >0  An error occurred.

**8745 CONSEQUENCES OF ERRORS**

8746       Default.

**8747 APPLICATION USAGE**

8748       The **-u** option has value in prototyping non-blocking reads from FIFOs. The intent is to support  
8749       the following sequence:

8750       mkfifo foo

8751       cat -u foo > /dev/tty13 &

8752       cat -u > foo

8753       It is unspecified whether standard output is or is not buffered in the default case. This is  
8754       sometimes of interest when standard output is associated with a terminal, since buffering may  
8755       delay the output. The presence of the **-u** option guarantees that unbuffered I/O is available. It is  
8756       implementation-defined whether the *cat* utility buffers output if the **-u** option is not specified.  
8757       Traditionally, the **-u** option is implemented using the equivalent of the *setvbuf()* function  
8758       defined in the System Interfaces volume of IEEE Std 1003.1-2001.

**8759 EXAMPLES**

8760       The following command:

8761       cat myfile

8762       writes the contents of the file **myfile** to standard output.

8763       The following command:

8764       cat doc1 doc2 > doc.all

8765       concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

8766       Because of the shell language mechanism used to perform output redirection, a command such  
8767       as this:

8768       cat doc doc.end > doc

8769       causes the original data in **doc** to be lost.

8770       The command:

8771       cat start - middle - end > file

when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a single invocation of *cat*. Note, however, that if standard input is a regular file, this would be equivalent to the command:

```
cat start - middle /dev/null end > file
```

because the entire contents of the file would be consumed by *cat* the first time '-' was used as a file operand and an end-of-file condition would be detected immediately when '-' was referenced the second time.

#### 8779 RATIONALE

Historical versions of the *cat* utility include the options **-e**, **-t**, and **-v**, which permit the ends of lines, <tab>s, and invisible characters, respectively, to be rendered visible in the output. The standard developers omitted these options because they provide too fine a degree of control over what is made visible, and similar output can be obtained using a command such as:

```
sed -n -e 's/$/$/ ' -e l pathname
```

The **-s** option was omitted because it corresponds to different functions in BSD and System V-based systems. The BSD **-s** option to squeeze blank lines can be accomplished by the shell script shown in the following example:

```
sed -n '
Write non-empty lines.
./ {
 p
 d
}
Write a single empty line, then look for more empty lines.
/^$/ p
Get next line, discard the held <newline> (empty line),
and look for more empty lines.
:Empty
/^$/ {
 N
 s/./ /
 b Empty
}
Write the non-empty line before going back to search
for the first in a set of empty lines.
p
'
```

The System V **-s** option to silence error messages can be accomplished by redirecting the standard error. Note that the BSD documentation for *cat* uses the term "blank line" to mean the same as the POSIX "empty line": a line consisting only of a <newline>.

The BSD **-n** option was omitted because similar functionality can be obtained from the **-n** option of the *pr* utility.

#### 8813 FUTURE DIRECTIONS

8814 None.

#### 8815 SEE ALSO

8816 *more*, the System Interfaces volume of IEEE Std 1003.1-2001, *setvbuf()*

8817 **CHANGE HISTORY**

8818 First released in Issue 2.

8819 **NAME**

8820           cd — change the working directory

8821 **SYNOPSIS**8822           cd [-L | -P] [*directory*]

8823           cd -

8824 **DESCRIPTION**

8825           The *cd* utility shall change the working directory of the current shell execution environment (see  
 8826           Section 2.12 (on page 61)) by executing the following steps in sequence. (In the following steps,  
 8827           the symbol **curpath** represents an intermediate value used to simplify the description of the  
 8828           algorithm used by *cd*. There is no requirement that **curpath** be made visible to the application.)

- 8829           1. If no *directory* operand is given and the *HOME* environment variable is empty or  
 8830           undefined, the default behavior is implementation-defined and no further steps shall be  
 8831           taken.
- 8832           2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty  
 8833           value, the *cd* utility shall behave as if the directory named in the *HOME* environment  
 8834           variable was specified as the *directory* operand.
- 8835           3. If the *directory* operand begins with a slash character, set **curpath** to the operand and  
 8836           proceed to step 7.
- 8837           4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.
- 8838           5. Starting with the first pathname in the colon-separated pathnames of *CDPATH* (see the  
 8839           ENVIRONMENT VARIABLES section) if the pathname is non-null, test if the  
 8840           concatenation of that pathname, a slash character, and the *directory* operand names a  
 8841           directory. If the pathname is null, test if the concatenation of dot, a slash character, and the  
 8842           operand names a directory. In either case, if the resulting string names an existing  
 8843           directory, set **curpath** to that string and proceed to step 7. Otherwise, repeat this step with  
 8844           the next pathname in *CDPATH* until all pathnames have been tested.
- 8845           6. Set **curpath** to the string formed by the concatenation of the value of *PWD*, a slash  
 8846           character, and the operand.
- 8847           7. If the **-P** option is in effect, the *cd* utility shall perform actions equivalent to the *chdir()*  
 8848           function, called with **curpath** as the *path* argument. If these actions succeed, the *PWD*  
 8849           environment variable shall be set to an absolute pathname for the current working  
 8850           directory and shall not contain filename components that, in the context of pathname  
 8851           resolution, refer to a file of type symbolic link. If there is insufficient permission on the new  
 8852           directory, or on any parent of that directory, to determine the current working directory,  
 8853           the value of the *PWD* environment variable is unspecified. If the actions equivalent to  
 8854           *chdir()* fail for any reason, the *cd* utility shall display an appropriate error message and not  
 8855           alter the *PWD* environment variable. Whether the actions equivalent to *chdir()* succeed or  
 8856           fail, no further steps shall be taken.
- 8857           8. The **curpath** value shall then be converted to canonical form as follows, considering each  
 8858           component from beginning to end, in sequence:
  - 8859           a. Dot components and any slashes that separate them from the next component shall  
 8860           be deleted.
  - 8861           b. For each dot-dot component, if there is a preceding component and it is neither root  
 8862           nor dot-dot, the preceding component, all slashes separating the preceding  
 8863           component from dot-dot, dot-dot and all slashes separating dot-dot from the  
 8864           following component shall be deleted.

8865 c. An implementation may further simplify **curpath** by removing any trailing slash  
 8866 characters that are not also leading slashes, replacing multiple non-leading  
 8867 consecutive slashes with a single slash, and replacing three or more leading slashes  
 8868 with a single slash. If, as a result of this canonicalization, the **curpath** variable is null,  
 8869 no further steps shall be taken.

8870 9. The *cd* utility shall then perform actions equivalent to the *chdir()* function called with  
 8871 **curpath** as the *path* argument. If these actions failed for any reason, the *cd* utility shall  
 8872 display an appropriate error message and no further steps shall be taken. The *PWD*  
 8873 environment variable shall be set to **curpath**.

8874 If, during the execution of the above steps, the *PWD* environment variable is changed, the  
 8875 *OLDPWD* environment variable shall also be changed to the value of the old working directory  
 8876 (that is the current working directory immediately prior to the call to *cd*).

## 8877 OPTIONS

8878 The *cd* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 8879 Utility Syntax Guidelines.

8880 The following options shall be supported by the implementation:

8881 **-L** Handle the operand dot-dot logically; symbolic link components shall not be  
 8882 resolved before dot-dot components are processed (see steps 8. and 9. in the  
 8883 DESCRIPTION).

8884 **-P** Handle the operand dot-dot physically; symbolic link components shall be  
 8885 resolved before dot-dot components are processed (see step 7. in the  
 8886 DESCRIPTION).

8887 If both **-L** and **-P** options are specified, the last of these options shall be used and all others  
 8888 ignored. If neither **-L** nor **-P** is specified, the operand shall be handled dot-dot logically; see the  
 8889 DESCRIPTION.

## 8890 OPERANDS

8891 The following operands shall be supported:

8892 *directory* An absolute or relative pathname of the directory that shall become the new  
 8893 working directory. The interpretation of a relative pathname by *cd* depends on the  
 8894 **-L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an  
 8895 empty string, the results are unspecified.

8896 **-** When a hyphen is used as the operand, this shall be equivalent to the command:  
 8897 `cd "$OLDPWD" && pwd`  
 8898 which changes to the previous working directory and then writes its name.

## 8899 STDIN

8900 Not used.

## 8901 INPUT FILES

8902 None.

## 8903 ENVIRONMENT VARIABLES

8904 The following environment variables shall affect the execution of *cd*:

8905 *CDPATH* A colon-separated list of pathnames that refer to directories. The *cd* utility shall use  
 8906 this list in its attempt to change the directory, as described in the DESCRIPTION.  
 8907 An empty string in place of a directory pathname represents the current directory.  
 8908 If *CDPATH* is not set, it shall be treated as if it were an empty string.

|      |                               |                                                                                                              |
|------|-------------------------------|--------------------------------------------------------------------------------------------------------------|
| 8909 | <i>HOME</i>                   | The name of the directory, used when no <i>directory</i> operand is specified.                               |
| 8910 | <i>LANG</i>                   | Provide a default value for the internationalization variables that are unset or null.                       |
| 8911 |                               | (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,                                       |
| 8912 |                               | Internationalization Variables for the precedence of internationalization variables                          |
| 8913 |                               | used to determine the values of locale categories.)                                                          |
| 8914 | <i>LC_ALL</i>                 | If set to a non-empty string value, override the values of all the other                                     |
| 8915 |                               | internationalization variables.                                                                              |
| 8916 | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as                            |
| 8917 |                               | characters (for example, single-byte as opposed to multi-byte characters in                                  |
| 8918 |                               | arguments).                                                                                                  |
| 8919 | <i>LC_MESSAGES</i>            |                                                                                                              |
| 8920 |                               | Determine the locale that should be used to affect the format and contents of                                |
| 8921 |                               | diagnostic messages written to standard error.                                                               |
| 8922 | XSI <i>NLSPATH</i>            | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                        |
| 8923 | <i>OLDPWD</i>                 | A pathname of the previous working directory, used by <i>cd -</i> .                                          |
| 8924 | <i>PWD</i>                    | This variable shall be set as specified in the DESCRIPTION. If an application sets                           |
| 8925 |                               | or unsets the value of <i>PWD</i> , the behavior of <i>cd</i> is unspecified.                                |
| 8926 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                              |
| 8927 |                               | Default.                                                                                                     |
| 8928 | <b>STDOUT</b>                 |                                                                                                              |
| 8929 |                               | If a non-empty directory name from <i>CDPATH</i> is used, or if <i>cd -</i> is used, an absolute pathname of |
| 8930 |                               | the new working directory shall be written to the standard output as follows:                                |
| 8931 |                               | <code>"%s\n", &lt;new directory&gt;</code>                                                                   |
| 8932 |                               | Otherwise, there shall be no output.                                                                         |
| 8933 | <b>STDERR</b>                 |                                                                                                              |
| 8934 |                               | The standard error shall be used only for diagnostic messages.                                               |
| 8935 | <b>OUTPUT FILES</b>           |                                                                                                              |
| 8936 |                               | None.                                                                                                        |
| 8937 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                              |
| 8938 |                               | None.                                                                                                        |
| 8939 | <b>EXIT STATUS</b>            |                                                                                                              |
| 8940 |                               | The following exit values shall be returned:                                                                 |
| 8941 | 0                             | The directory was successfully changed.                                                                      |
| 8942 | >0                            | An error occurred.                                                                                           |
| 8943 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                              |
| 8944 |                               | The working directory shall remain unchanged.                                                                |



**APPLICATION USAGE**

Since *cd* affects the current shell execution environment, it is always provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(cd /tmp)
nohup cd
find . -exec cd {} \;
```

it does not affect the working directory of the caller's environment.

The user must have execute (search) permission in *directory* in order to change to it.

**EXAMPLES**

None.

**RATIONALE**

The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.

A common extension when *HOME* is undefined is to get the login directory from the user database for the invoking user. This does not occur on System V implementations.

Some historical shells, such as the KornShell, took special actions when the directory name contained a dot-dot component, selecting the logical parent of the directory, rather than the actual parent directory; that is, it moved up one level toward the '/' in the pathname, remembering what the user typed, rather than performing the equivalent of:

```
chdir("../");
```

In such a shell, the following commands would not necessarily produce equivalent output for all directories:

```
cd .. && ls ls ..
```

This behavior is now the default. It is not consistent with the definition of dot-dot in most historical practice; that is, while this behavior has been optionally available in the KornShell, other shells have historically not supported this functionality. The logical pathname is stored in the *PWD* environment variable when the *cd* utility completes and this value is used to construct the next directory name if *cd* is invoked with the *-L* option.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

Section 2.12 (on page 61), *pwd*, the System Interfaces volume of IEEE Std 1003.1-2001, *chdir()*

**CHANGE HISTORY**

First released in Issue 2.

**Issue 6**

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *cd* – operand, *PWD*, and *OLDPWD* are added.

The *-L* and *-P* options are added to align with the IEEE P1003.2b draft standard. This also includes the introduction of a new description to include the effect of these options.

8986 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/14 is applied, changing the SYNOPSIS to |  
8987 make it clear that the **-L** and **-P** options are mutually-exclusive. |

8988 **NAME**8989 cflow — generate a C-language flowgraph (**DEVELOPMENT**)8990 **SYNOPSIS**

```
8991 xSI cflow [-r] [-d num] [-D name[=def]] ... [-i incl] [-I dir] ...
8992 [-U dir] ... file ...
```

8993

8994 **DESCRIPTION**

8995 The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex*, or *yacc*  
 8996 source files, and attempt to build a graph, written to standard output, charting the external  
 8997 references.

8998 **OPTIONS**

8999 The *cflow* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 9000 12.2, Utility Syntax Guidelines, except that the order of the **-D**, **-I**, and **-U** options (which are  
 9001 identical to their interpretation by *c99*) is significant.

9002 The following options shall be supported:

9003 **-d num** Indicate the depth at which the flowgraph is cut off. The application shall ensure  
 9004 that the argument *num* is a decimal integer. By default this is a very large number  
 9005 (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive  
 9006 integer shall be ignored.

9007 **-i incl** Increase the number of included symbols. The *incl* option-argument is one of the  
 9008 following characters:

9009 *x* Include external and static data symbols. The default shall be to include only  
 9010 functions in the flowgraph.

9011 *\_* (Underscore) Include names that begin with an underscore. The default shall  
 9012 be to exclude these functions (and data if **-i x** is used).

9013 **-r** Reverse the caller: callee relationship, producing an inverted listing showing the  
 9014 callers of each function. The listing shall also be sorted in lexicographical order by  
 9015 callee.

9016 **OPERANDS**

9017 The following operand is supported:

9018 *file* The pathname of a file for which a graph is to be generated. Filenames suffixed by  
 9019 **.l** shall be taken to be *lex* input, **.y** as *yacc* input, **.c** as *c99* input, and **.i** as the  
 9020 output of *c99 -E*. Such files shall be processed as appropriate, determined by their  
 9021 suffix.

9022 Files suffixed by **.s** (conventionally assembler source) may have more limited  
 9023 information extracted from them.

9024 **STDIN**

9025 Not used.

9026 **INPUT FILES**

9027 The input files shall be object files or assembler, C-language, *lex*, or *yacc* source files.

9028 **ENVIRONMENT VARIABLES**

9029 The following environment variables shall affect the execution of *cflow*:

9030 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 9031 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,

9032 Internationalization Variables for the precedence of internationalization variables  
 9033 used to determine the values of locale categories.)

9034 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 9035 internationalization variables.

9036 *LC\_COLLATE*  
 9037 Determine the locale for the ordering of the output when the *-r* option is used.

9038 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 9039 characters (for example, single-byte as opposed to multi-byte characters in  
 9040 arguments and input files).

9041 *LC\_MESSAGES*  
 9042 Determine the locale that should be used to affect the format and contents of  
 9043 diagnostic messages written to standard error.

9044 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

9045 **ASYNCHRONOUS EVENTS**  
 9046 Default.

9047 **STDOUT**  
 9048 The flowgraph written to standard output shall be formatted as follows:  
 9049 "%d %s:%s\n", <reference number>, <global>, <definition>

9050 Each line of output begins with a reference (that is, line) number, followed by indentation of at  
 9051 least one column position per level. This is followed by the name of the global, a colon, and its  
 9052 definition. Normally globals are only functions not defined as an external or beginning with an  
 9053 underscore; see the **OPTIONS** section for the *-i* inclusion option. For information extracted from  
 9054 C-language source, the definition consists of an abstract type declaration (for example, **char \***)  
 9055 and, delimited by angle brackets, the name of the source file and the line number where the  
 9056 definition was found. Definitions extracted from object files indicate the filename and location  
 9057 counter under which the symbol appeared (for example, *text*).

9058 Once a definition of a name has been written, subsequent references to that name contain only  
 9059 the reference number of the line where the definition can be found. For undefined references,  
 9060 only "< >" shall be written.

9061 **STDERR**  
 9062 The standard error shall be used only for diagnostic messages.

9063 **OUTPUT FILES**  
 9064 None.

9065 **EXTENDED DESCRIPTION**  
 9066 None.

9067 **EXIT STATUS**  
 9068 The following exit values shall be returned:  
 9069 0 Successful completion.  
 9070 >0 An error occurred.

9071 **CONSEQUENCES OF ERRORS**  
 9072 Default.

**9073 APPLICATION USAGE**

9074 Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can  
9075 confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

**9076 EXAMPLES**

9077 Given the following in **file.c**:

```
9078 int i;
9079 int f();
9080 int g();
9081 int h();
9082 int
9083 main()
9084 {
9085 f();
9086 g();
9087 f();
9088 }
9089 int
9090 f()
9091 {
9092 i = h();
9093 }
```

9094 The command:

9095 `cflow -i x file.c`

9096 produces the output:

```
9097 1 main: int(), <file.c 6>
9098 2 f: int(), <file.c 13>
9099 3 h: <>
9100 4 i: int, <file.c 1>
9101 5 g: <>
```

**9102 RATIONALE**

9103 None.

**9104 FUTURE DIRECTIONS**

9105 None.

**9106 SEE ALSO**

9107 *c99*, *lex*, *yacc*

**9108 CHANGE HISTORY**

9109 First released in Issue 2.

**9110 Issue 6**

9111 The normative text is reworded to avoid use of the term “must” for application requirements.

9112 **NAME**

9113 chgrp — change the file group ownership

9114 **SYNOPSIS**9115 chgrp [-hR] *group file ...* |9116 chgrp -R [-H | -L | -P ] *group file ...* |9117 **DESCRIPTION**9118 The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID  
9119 specified by the *group* operand.9120 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the  
9121 directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to  
9122 the *chown()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001, called  
9123 with the following arguments:

- 9124 • The *file* operand shall be used as the *path* argument.
- 9125 • The user ID of the file shall be used as the *owner* argument.
- 9126 • The specified group ID shall be used as the *group* argument.

9127 Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-  
9128 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-  
9129 group-ID bits of other file types may be cleared.9130 **OPTIONS**9131 The *chgrp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
9132 12.2, Utility Syntax Guidelines.

9133 The following options shall be supported by the implementation:

- 9134 **-h** If the system supports group IDs for symbolic links, for each *file* operand that  
9135 names a file of type symbolic link, *chgrp* shall attempt to set the group ID of the  
9136 symbolic link instead of the file referenced by the symbolic link. If the system does  
9137 not support group IDs for symbolic links, for each *file* operand that names a file of  
9138 type symbolic link, *chgrp* shall do nothing more with the current file and shall go  
9139 on to any remaining files.
- 9140 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory  
9141 is specified on the command line, *chgrp* shall change the group of the directory  
9142 referenced by the symbolic link and all files in the file hierarchy below it.
- 9143 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory  
9144 is specified on the command line or encountered during the traversal of a file  
9145 hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic  
9146 link and all files in the file hierarchy below it.
- 9147 **-P** If the **-R** option is specified and a symbolic link is specified on the command line  
9148 or encountered during the traversal of a file hierarchy, *chgrp* shall change the  
9149 group ID of the symbolic link if the system supports this operation. The *chgrp*  
9150 utility shall not follow the symbolic link to any other part of the file hierarchy.
- 9151 **-R** Recursively change file group IDs. For each *file* operand that names a directory,  
9152 *chgrp* shall change the group of the directory and all files in the file hierarchy below  
9153 it. Unless a **-H**, **-L**, or **-P** option is specified, it is unspecified which of these  
9154 options will be used as the default.

9155 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be  
 9156 considered an error. The last option specified shall determine the behavior of the utility.

## 9157 OPERANDS

9158 The following operands shall be supported:

9159 *group* A group name from the group database or a numeric group ID. Either specifies a  
 9160 group ID to be given to each file named by one of the *file* operands. If a numeric  
 9161 *group* operand exists in the group database as a group name, the group ID number  
 9162 associated with that group name is used as the group ID.

9163 *file* A pathname of a file whose group ID is to be modified.

## 9164 STDIN

9165 Not used.

## 9166 INPUT FILES

9167 None.

## 9168 ENVIRONMENT VARIABLES

9169 The following environment variables shall affect the execution of *chgrp*:

9170 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 9171 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 9172 Internationalization Variables for the precedence of internationalization variables  
 9173 used to determine the values of locale categories.)

9174 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 9175 internationalization variables.

9176 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 9177 characters (for example, single-byte as opposed to multi-byte characters in  
 9178 arguments).

9179 *LC\_MESSAGES*

9180 Determine the locale that should be used to affect the format and contents of  
 9181 diagnostic messages written to standard error.

9182 *NSI* *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 9183 ASYNCHRONOUS EVENTS

9184 Default.

## 9185 STDOUT

9186 Not used.

## 9187 STDERR

9188 The standard error shall be used only for diagnostic messages.

## 9189 OUTPUT FILES

9190 None.

## 9191 EXTENDED DESCRIPTION

9192 None.

## 9193 EXIT STATUS

9194 The following exit values shall be returned:

9195 0 The utility executed successfully and all requested changes were made.

9196 >0 An error occurred.

9197 **CONSEQUENCES OF ERRORS**

9198 Default.

9199 **APPLICATION USAGE**9200 Only the owner of a file or the user with appropriate privileges may change the owner or group  
9201 of a file.9202 Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the  
9203 *group* specified is not the effective group ID or one of the supplementary group IDs of the calling  
9204 process.9205 **EXAMPLES**

9206 None.

9207 **RATIONALE**9208 The System V and BSD versions use different exit status codes. Some implementations used the  
9209 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
9210 can overflow the range of valid exit status values. The standard developers chose to mask these  
9211 by specifying only 0 and >0 as exit values.9212 The functionality of *chgrp* is described substantially through references to *chown()*. In this way,  
9213 there is no duplication of effort required for describing the interactions of permissions, multiple  
9214 groups, and so on.9215 **FUTURE DIRECTIONS**

9216 None.

9217 **SEE ALSO**9218 *chmod*, *chown*, the System Interfaces volume of IEEE Std 1003.1-2001, *chown()*9219 **CHANGE HISTORY**

9220 First released in Issue 2.

9221 **Issue 6**9222 New options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
9223 options affect the processing of symbolic links.9224 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
9225 section to “Default.”.9226 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/15 is applied, changing the SYNOPSIS to  
9227 make it clear that **-h** and **-R** are optional. |



9228 **NAME**

9229           chmod — change the file modes

9230 **SYNOPSIS**9231           chmod [-R] *mode file ...*9232 **DESCRIPTION**9233           The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*  
9234           operand in the way specified by the *mode* operand.9235           It is implementation-defined whether and how the *chmod* utility affects any alternate or  
9236           additional file access control mechanism (see the Base Definitions volume of  
9237           IEEE Std 1003.1-2001, Section 4.4, File Access Permissions) being used for the specified file.9238           Only a process whose effective user ID matches the user ID of the file, or a process with the  
9239           appropriate privileges, shall be permitted to change the file mode bits of a file.9240 **OPTIONS**9241           The *chmod* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
9242           12.2, Utility Syntax Guidelines.

9243           The following option shall be supported:

9244           **-R**           Recursively change file mode bits. For each *file* operand that names a directory,  
9245           *chmod* shall change the file mode bits of the directory and all files in the file  
9246           hierarchy below it.9247 **OPERANDS**

9248           The following operands shall be supported:

9249           *mode*           Represents the change to be made to the file mode bits of each file named by one of  
9250           the *file* operands; see the EXTENDED DESCRIPTION section.9251           *file*           A pathname of a file whose file mode bits shall be modified.9252 **STDIN**

9253           Not used.

9254 **INPUT FILES**

9255           None.

9256 **ENVIRONMENT VARIABLES**9257           The following environment variables shall affect the execution of *chmod*:9258           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
9259           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
9260           Internationalization Variables for the precedence of internationalization variables  
9261           used to determine the values of locale categories.)9262           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
9263           internationalization variables.9264           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
9265           characters (for example, single-byte as opposed to multi-byte characters in  
9266           arguments).9267           **LC\_MESSAGES**9268                           Determine the locale that should be used to affect the format and contents of  
9269           diagnostic messages written to standard error.

9270 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

9271 **ASYNCHRONOUS EVENTS**

9272 Default.

9273 **STDOUT**

9274 Not used.

9275 **STDERR**

9276 The standard error shall be used only for diagnostic messages.

9277 **OUTPUT FILES**

9278 None.

9279 **EXTENDED DESCRIPTION**

9280 The *mode* operand shall be either a *symbolic\_mode* expression or a non-negative octal integer. The

9281 *symbolic\_mode* form is described by the grammar later in this section.

9282 Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*.

9283 The operations shall be performed on each *file* in the order in which the **clauses** are specified.

9284 The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits,

9285 respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.

9286 The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute/search* portions of file mode

9287 bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who**

9288 contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.

9289 The **perm** symbol **X** shall represent the execute/search portion of the file mode bits if the file is a

9290 directory or if the current (unmodified) file mode bits have at least one of the execute bits

9291 (S\_IXUSR, S\_IXGRP, or S\_IXOTH) set. It shall be ignored if the file is not a directory and none of

9292 the execute bits are set in the current file mode bits.

9293 The **permcop** symbols **u**, **g**, and **o** shall represent the current permissions associated with the

9294 user, group, and other parts of the file mode bits, respectively. For the remainder of this section,

9295 **perm** refers to the non-terminals **perm** and **permcop** in the grammar.

9296 If multiple **actionlists** are grouped with a single **wholist** in the grammar, each **actionlist** shall be

9297 applied in the order specified with that **wholist**. The *op* symbols shall represent the operation

9298 performed, as follows:

9299 + If **perm** is not specified, the '+' operation shall not change the file mode bits.

9300 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and

9301 other permissions, except for those with corresponding bits in the file mode creation mask

9302 of the invoking process, shall be set.

9303 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

9304 – If **perm** is not specified, the '-' operation shall not change the file mode bits.

9305 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and

9306 other permissions, except for those with corresponding bits in the file mode creation mask

9307 of the invoking process, shall be cleared.

9308 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be

9309 cleared.

9310 = Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the

9311 file mode bits specified in this volume of IEEE Std 1003.1-2001.

If **perm** is not specified, the '=' operation shall make no further modifications to the file mode bits.

If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be set.

Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

When using the symbolic mode form on a regular file, it is implementation-defined whether or not:

- Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all execute bits are currently clear and none are being set are ignored.
- Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-on-execution bits.
- Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all execute bits are currently clear are ignored. However, if the command `ls -l file` writes an `s` in the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is set, the commands `chmod u-s file` or `chmod g-s file`, respectively, shall not be ignored.

When using the symbolic mode form on other file types, it is implementation-defined whether or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are honored.

If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm** symbol **s**.

The **perm** symbol **t** shall specify the `S_ISVTX` bit. When used with a file of type directory, it can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other than directory is unspecified.

For an octal integer *mode* operand, the file mode bits shall be set absolutely.

For each bit set in the octal number, the corresponding file permission bit shown in the following table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-execution, bits shown in the following table shall be set; if these bits are not set in the octal number, they are cleared. For other file types, it is implementation-defined whether or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are honored.

| Octal | Mode Bit | Octal | Mode Bit | Octal | Mode Bit | Octal | Mode Bit |
|-------|----------|-------|----------|-------|----------|-------|----------|
| 4000  | S_ISUID  | 0400  | S_IRUSR  | 0040  | S_IRGRP  | 0004  | S_IROTH  |
| 2000  | S_ISGID  | 0200  | S_IWUSR  | 0020  | S_IWGRP  | 0002  | S_IWOTH  |
| 1000  | S_ISVTX  | 0100  | S_IXUSR  | 0010  | S_IXGRP  | 0001  | S_IXOTH  |

When bits are set in the octal number other than those listed in the table above, the behavior is unspecified.

9354 **Grammar for chmod**

9355 The grammar and lexical conventions in this section describe the syntax for the *symbolic\_mode*  
 9356 operand. The general conventions for this style of grammar are described in Section 1.10 (on  
 9357 page 19). A valid *symbolic\_mode* can be represented as the non-terminal symbol *symbolic\_mode* in  
 9358 the grammar. This formal syntax shall take precedence over the preceding text syntax  
 9359 description.

9360 The lexical processing is based entirely on single characters. Implementations need not allow  
 9361 <blank>s within the single argument being processed.

9362 %start symbolic\_mode  
 9363 %%

9364 symbolic\_mode : clause  
 9365 | symbolic\_mode ',' clause  
 9366 ;

9367 clause : actionlist  
 9368 | wholist actionlist  
 9369 ;

9370 wholist : who  
 9371 | wholist who  
 9372 ;

9373 who : 'u' | 'g' | 'o' | 'a'  
 9374 ;

9375 actionlist : action  
 9376 | actionlist action  
 9377 ;

9378 action : op  
 9379 | op permlist  
 9380 | op permcopy  
 9381 ;

9382 permcopy : 'u' | 'g' | 'o'  
 9383 ;

9384 op : '+' | '-' | '='  
 9385 ;

9386 permlist : perm  
 9387 | perm permlist  
 9388 ;

9389 xSI perm : 'r' | 'w' | 'x' | 'X' | 's' | 't'  
 9390 ;

9391 **EXIT STATUS**

9392 The following exit values shall be returned:

9393 0 The utility executed successfully and all requested changes were made.

9394 >0 An error occurred.

9395 **CONSEQUENCES OF ERRORS**

9396 Default.

9397 **APPLICATION USAGE**

9398 Some implementations of the *chmod* utility change the mode of a directory before the files in the  
 9399 directory when performing a recursive (**-R** option) change; others change the directory mode  
 9400 after the files in the directory. If an application tries to remove read or search permission for a  
 9401 file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying  
 9402 to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users  
 9403 should not try to make a hierarchy inaccessible to themselves.

9404 Some implementations of *chmod* never used the process' *umask* when changing modes; systems  
 9405 conformant with this volume of IEEE Std 1003.1-2001 do so when **who** is not specified. Note the  
 9406 difference between:

9407 `chmod a-w file`

9408 which removes all write permissions, and:

9409 `chmod -- -w file`

9410 which removes write permissions that would be allowed if **file** was created with the same  
 9411 *umask*.

9412 Conforming applications should never assume that they know how the set-user-ID and set-  
 9413 group-ID bits on directories are interpreted.

9414 **EXAMPLES**

| Mode         | Results                                                                                            |
|--------------|----------------------------------------------------------------------------------------------------|
| <i>a+=</i>   | Equivalent to <i>a+,a=</i> ; clears all file mode bits.                                            |
| <i>go+-w</i> | Equivalent to <i>go+,go-w</i> ; clears group and other write bits.                                 |
| <i>g=o-w</i> | Equivalent to <i>g=o,g-w</i> ; sets group bit to match other bits and then clears group write bit. |
| <i>g-r+w</i> | Equivalent to <i>g-r,g+w</i> ; clears group read bit and sets group write bit.                     |
| <i>uo=g</i>  | Sets owner bits to match group bits and sets other bits to match group bits.                       |

9425 **RATIONALE**

9426 The functionality of *chmod* is described substantially through references to concepts defined in  
 9427 the System Interfaces volume of IEEE Std 1003.1-2001. In this way, there is less duplication of  
 9428 effort required for describing the interactions of permissions. However, the behavior of this  
 9429 utility is not described in terms of the *chmod()* function from the System Interfaces volume of  
 9430 IEEE Std 1003.1-2001 because that specification requires certain side effects upon alternate file  
 9431 access control mechanisms that might not be appropriate, depending on the implementation.

9432 Implementations that support mandatory file and record locking as specified by the 1984  
 9433 /usr/group standard historically used the combination of set-group-ID bit set and group  
 9434 execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the  
 9435 symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory  
 9436 locking mode is not changed without explicit indication that that was what the user intended.  
 9437 Therefore, the details on how the implementation treats these conditions must be defined in the  
 9438 documentation. This volume of IEEE Std 1003.1-2001 does not require mandatory locking (nor  
 9439 does the System Interfaces volume of IEEE Std 1003.1-2001), but does allow it as an extension.  
 9440 However, this volume of IEEE Std 1003.1-2001 does require that the *ls* and *chmod* utilities work

consistently in this area. If *ls -l file* indicates that the set-group-ID bit is set, *chmod g-s file* must clear it (assuming appropriate privileges exist to change modes).

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. This problem is avoided here by specifying only 0 and >0 as exit values.

The System Interfaces volume of IEEE Std 1003.1-2001 indicates that implementation-defined restrictions may cause the S\_ISUID and S\_ISGID bits to be ignored. This volume of IEEE Std 1003.1-2001 allows the *chmod* utility to choose to modify these bits before calling *chmod()* (or some function providing equivalent capabilities) for non-regular files. Among other things, this allows implementations that use the set-user-ID and set-group-ID bits on directories to enable extended features to handle these extensions in an intelligent manner.

The **X perm** symbol was adopted from BSD-based systems because it provides commonly desired functionality when doing recursive (**-R** option) modifications. Similar functionality is not provided by the *find* utility. Historical BSD versions of *chmod*, however, only supported **X** with *op+*; it has been extended in this volume of IEEE Std 1003.1-2001 because it is also useful with *op=*. (It has also been added for *op-* even though it duplicates **x**, in this case, because it is intuitive and easier to explain.)

The grammar was extended with the *permcop* non-terminal to allow historical-practice forms of symbolic modes like **o=u-g** (that is, set the “other” permissions to the permissions of “owner” minus the permissions of “group”).

#### 9462 FUTURE DIRECTIONS

9463 None.

#### 9464 SEE ALSO

9465 *ls*, *umask*, the System Interfaces volume of IEEE Std 1003.1-2001, *chmod()*

#### 9466 CHANGE HISTORY

9467 First released in Issue 2.

#### 9468 Issue 6

9469 The following new requirements on POSIX implementations derive from alignment with the  
9470 Single UNIX Specification:

- 9471 • Octal modes have been kept and made mandatory despite being marked obsolescent in the  
9472 ISO POSIX-2: 1993 standard.

9473 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
9474 section to “Default.”.

9475 The Open Group Base Resolution bwg2001-010 is applied, adding the description of the  
9476 S\_ISVTX bit and the **t perm** symbol as an XSI extension.

9477 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/16 is applied, changing the XSI shaded  
9478 text in the EXTENDED DESCRIPTION from:

9479 “The **perm** symbol **t** shall specify the S\_ISVTX bit and shall apply to directories only. The  
9480 effect when using it with any other file type is unspecified. It can be used with the **who**  
9481 symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u** or  
9482 **g** in conjunction with the **perm** symbol **t**; it shall be ignored for **u** and **g**.”

9483 to:

9484            “The **perm** symbol **t** shall specify the S\_ISVTX bit. When used with a file of type directory, it  
9485            can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify  
9486            a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these  
9487            combinations is unspecified. The effect when using the **perm** symbol **t** with any file type  
9488            other than directory is unspecified.”

9489            This change is to permit historical behavior.

9490 **NAME**

9491 chown — change the file ownership

9492 **SYNOPSIS**

9493 chown [-hR] owner[:group] file ... |

9494 chown -R [-H | -L | -P ] owner[:group] file ... |

9495 **DESCRIPTION**9496 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID  
9497 specified by the *owner* operand.9498 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the  
9499 directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to  
9500 the *chown()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001, called  
9501 with the following arguments:

- 9502 1. The *file* operand shall be used as the *path* argument.
- 9503 2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner*  
9504 argument.
- 9505 3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used  
9506 as the *group* argument; otherwise, the group ownership shall not be changed.

9507 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-  
9508 group-ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and  
9509 set-group-ID bits of other file types may be cleared.9510 **OPTIONS**9511 The *chown* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
9512 12.2, Utility Syntax Guidelines.

9513 The following options shall be supported by the implementation:

9514 **-h** If the system supports user IDs for symbolic links, for each *file* operand that names  
9515 a file of type symbolic link, *chown* shall attempt to set the user ID of the symbolic  
9516 link. If the system supports group IDs for symbolic links, and a group ID was  
9517 specified, for each *file* operand that names a file of type symbolic link, *chown* shall  
9518 attempt to set the group ID of the symbolic link. If the system does not support  
9519 user or group IDs for symbolic links, for each *file* operand that names a file of type  
9520 symbolic link, *chown* shall do nothing more with the current file and shall go on to  
9521 any remaining files.

9522 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory  
9523 is specified on the command line, *chown* shall change the user ID (and group ID, if  
9524 specified) of the directory referenced by the symbolic link and all files in the file  
9525 hierarchy below it.

9526 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory  
9527 is specified on the command line or encountered during the traversal of a file  
9528 hierarchy, *chown* shall change the user ID (and group ID, if specified) of the  
9529 directory referenced by the symbolic link and all files in the file hierarchy below it.

9530 **-P** If the **-R** option is specified and a symbolic link is specified on the command line  
9531 or encountered during the traversal of a file hierarchy, *chown* shall change the  
9532 owner ID (and group ID, if specified) of the symbolic link if the system supports  
9533 this operation. The *chown* utility shall not follow the symbolic link to any other  
9534 part of the file hierarchy.



9535           **-R**           Recursively change file user and group IDs. For each *file* operand that names a  
 9536           directory, *chown* shall change the user ID (and group ID, if specified) of the  
 9537           directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is  
 9538           specified, it is unspecified which of these options will be used as the default.

9539           Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be  
 9540           considered an error. The last option specified shall determine the behavior of the utility.

#### 9541 **OPERANDS**

9542           The following operands shall be supported:

9543           **owner[:group]** A user ID and optional group ID to be assigned to *file*. The *owner* portion of this  
 9544           operand shall be a user name from the user database or a numeric user ID. Either  
 9545           specifies a user ID which shall be given to each file named by one of the *file*  
 9546           operands. If a numeric *owner* operand exists in the user database as a user name,  
 9547           the user ID number associated with that user name shall be used as the user ID.  
 9548           Similarly, if the *group* portion of this operand is present, it shall be a group name  
 9549           from the group database or a numeric group ID. Either specifies a group ID which  
 9550           shall be given to each file. If a numeric group operand exists in the group database  
 9551           as a group name, the group ID number associated with that group name shall be  
 9552           used as the group ID.

9553           *file*           A pathname of a file whose user ID is to be modified.

#### 9554 **STDIN**

9555           Not used.

#### 9556 **INPUT FILES**

9557           None.

#### 9558 **ENVIRONMENT VARIABLES**

9559           The following environment variables shall affect the execution of *chown*:

9560           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
 9561           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 9562           Internationalization Variables for the precedence of internationalization variables  
 9563           used to determine the values of locale categories.)

9564           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
 9565           internationalization variables.

9566           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
 9567           characters (for example, single-byte as opposed to multi-byte characters in  
 9568           arguments).

9569           **LC\_MESSAGES**

9570                               Determine the locale that should be used to affect the format and contents of  
 9571           diagnostic messages written to standard error.

9572   XSI       **NLSPATH**       Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

#### 9573 **ASYNCHRONOUS EVENTS**

9574           Default.

#### 9575 **STDOUT**

9576           Not used.

9577 **STDERR**

9578           The standard error shall be used only for diagnostic messages.

9579 **OUTPUT FILES**

9580           None.

9581 **EXTENDED DESCRIPTION**

9582           None.

9583 **EXIT STATUS**

9584           The following exit values shall be returned:

9585           0   The utility executed successfully and all requested changes were made.

9586           >0   An error occurred.

9587 **CONSEQUENCES OF ERRORS**

9588           Default.

9589 **APPLICATION USAGE**

9590           Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

9592           Some implementations restrict the use of *chown* to a user with appropriate privileges.

9593 **EXAMPLES**

9594           None.

9595 **RATIONALE**

9596           The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. These are masked by specifying only 0 and >0 as exit values.

9600           The functionality of *chown* is described substantially through references to functions in the System Interfaces volume of IEEE Std 1003.1-2001. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, and so on.

9603           The 4.3 BSD method of specifying both owner and group was included in this volume of IEEE Std 1003.1-2001 because:

9605           • There are cases where the desired end condition could not be achieved using the *chgrp* and *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the desired group and the desired owner is not a member of the current group, the *chown()* function could fail unless both owner and group are changed at the same time.)

9609           • Even if they could be changed independently, in cases where both are being changed, there is a 100% performance penalty caused by being forced to invoke both utilities.

9611           The BSD syntax *user[group]* was changed to *user[:group]* in this volume of IEEE Std 1003.1-2001 because the period is a valid character in login names (as specified by the Base Definitions volume of IEEE Std 1003.1-2001, login names consist of characters in the portable filename character set). The colon character was chosen as the replacement for the period character because it would never be allowed as a character in a user name or group name on historical implementations.

9617           The **-R** option is considered by some observers as an undesirable departure from the historical UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there seemed to be no good reason to require other tools to have to duplicate that functionality. However, the **-R** option was deemed an important user convenience, is far more efficient than

9621 forking a separate process for each element of the directory hierarchy, and is in widespread  
9622 historical use.

9623 **FUTURE DIRECTIONS**

9624 None.

9625 **SEE ALSO**

9626 *chmod*, *chgrp*, the System Interfaces volume of IEEE Std 1003.1-2001, *chown()*

9627 **CHANGE HISTORY**

9628 First released in Issue 2.

9629 **Issue 6**

9630 New options **-h**, **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
9631 options affect the processing of symbolic links.

9632 The normative text is reworded to avoid use of the term “must” for application requirements.

9633 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
9634 section to “Default.”.

9635 The “otherwise, ...” text in item 3. of the DESCRIPTION is changed to “otherwise, the group  
9636 ownership shall not be changed”.

9637 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/17 is applied, changing the SYNOPSIS to  
9638 make it clear that **-h** and **-R** are optional. |

9639 **NAME**

9640 cksum — write file checksums and sizes

9641 **SYNOPSIS**9642 cksum [*file* ...]9643 **DESCRIPTION**

9644 The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC)  
 9645 for each input file, and also write to standard output the number of octets in each file. The CRC  
 9646 used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996  
 9647 standard (Ethernet).

9648 The encoding for the CRC checksum is defined by the generating polynomial:

$$9649 G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

9650 Mathematically, the CRC value corresponding to a given file shall be defined by the following  
 9651 procedure:

- 9652 1. The *n* bits to be evaluated are considered to be the coefficients of a mod 2 polynomial *M*(*x*)  
 9653 of degree *n*−1. These *n* bits are the bits from the file, with the most significant bit being the  
 9654 most significant bit of the first octet of the file and the last bit being the least significant bit  
 9655 of the last octet, padded with zero bits (if necessary) to achieve an integral number of  
 9656 octets, followed by one or more octets representing the length of the file as a binary value,  
 9657 least significant octet first. The smallest number of octets capable of representing this  
 9658 integer shall be used.
- 9659 2. *M*(*x*) is multiplied by  $x^{32}$  (that is, shifted left 32 bits) and divided by *G*(*x*) using mod 2  
 9660 division, producing a remainder *R*(*x*) of degree ≤ 31.
- 9661 3. The coefficients of *R*(*x*) are considered to be a 32-bit sequence.
- 9662 4. The bit sequence is complemented and the result is the CRC.

9663 **OPTIONS**

9664 None.

9665 **OPERANDS**

9666 The following operand shall be supported:

9667 *file* A pathname of a file to be checked. If no *file* operands are specified, the standard  
 9668 input shall be used.

9669 **STDIN**

9670 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 9671 section.

9672 **INPUT FILES**

9673 The input files can be any file type.

9674 **ENVIRONMENT VARIABLES**9675 The following environment variables shall affect the execution of *cksum*:

9676 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 9677 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 9678 Internationalization Variables for the precedence of internationalization variables  
 9679 used to determine the values of locale categories.)

9680 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 9681 internationalization variables.

9682            *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 9683                                   characters (for example, single-byte as opposed to multi-byte characters in  
 9684                                   arguments).

9685            *LC\_MESSAGES*  
 9686                                   Determine the locale that should be used to affect the format and contents of  
 9687                                   diagnostic messages written to standard error.

9688   *XSI*        *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

9689   **ASYNCHRONOUS EVENTS**  
 9690                   Default.

9691   **STDOUT**  
 9692                   For each file processed successfully, the *cksum* utility shall write in the following format:  
 9693                   "%u %d %s\n", <checksum>, <# of octets>, <pathname>  
 9694                   If no *file* operand was specified, the pathname and its leading <space> shall be omitted.

9695   **STDERR**  
 9696                   The standard error shall be used only for diagnostic messages.

9697   **OUTPUT FILES**  
 9698                   None.

9699   **EXTENDED DESCRIPTION**  
 9700                   None.

9701   **EXIT STATUS**  
 9702                   The following exit values shall be returned:  
 9703                   0   All files were processed successfully.  
 9704                   >0   An error occurred.

9705   **CONSEQUENCES OF ERRORS**  
 9706                   Default.

9707   **APPLICATION USAGE**  
 9708                   The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of  
 9709                   the same, such as to ensure that files transmitted over noisy media arrive intact. However, this  
 9710                   comparison cannot be considered cryptographically secure. The chances of a damaged file  
 9711                   producing the same CRC as the original are small; deliberate deception is difficult, but probably  
 9712                   not impossible.

9713                   Although input files to *cksum* can be any type, the results need not be what would be expected  
 9714                   on character special device files or on file types not described by the System Interfaces volume of  
 9715                   IEEE Std 1003.1-2001. Since this volume of IEEE Std 1003.1-2001 does not specify the block size  
 9716                   used when doing input, checksums of character special files need not process all of the data in  
 9717                   those files.

9718                   The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted  
 9719                   between two systems and undergoes any data transformation (such as changing little-endian  
 9720                   byte ordering to big-endian), identical CRC values cannot be expected. Implementations  
 9721                   performing such transformations may extend *cksum* to handle such situations.

9722 **EXAMPLES**

9723       None.

9724 **RATIONALE**

9725       The following C-language program can be used as a model to describe the algorithm. It assumes  
 9726       that a **char** is one octet. It also assumes that the entire file is available for one pass through the  
 9727       function. This was done for simplicity in demonstrating the algorithm, rather than as an  
 9728       implementation model.

```

9729 static unsigned long crctab[] = {
9730 0x00000000,
9731 0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
9732 0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
9733 0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbd8d,
9734 0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
9735 0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
9736 0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
9737 0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
9738 0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
9739 0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
9740 0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
9741 0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
9742 0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1, 0xe13ef6f4,
9743 0xe5ffe643, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
9744 0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
9745 0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcd8bb16,
9746 0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
9747 0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93dddb, 0x6f52c06c,
9748 0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
9749 0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
9750 0xaca5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e, 0xbfa1b04b,
9751 0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
9752 0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
9753 0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
9754 0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
9755 0xc27dede8, 0xcf3ecb31, 0xcbffd686, 0xd5b88683, 0xd1799b34,
9756 0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcd5d59, 0x608edb80,
9757 0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
9758 0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
9759 0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
9760 0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
9761 0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
9762 0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
9763 0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
9764 0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
9765 0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
9766 0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
9767 0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9ff77d71,
9768 0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
9769 0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
9770 0x4e8ee645, 0x4a4fffb2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
9771 0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
9772 0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,

```

```

9773 0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
9774 0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
9775 0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
9776 0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
9777 0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xee2ed18,
9778 0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
9779 0x8d79e0be, 0x803ac667, 0x84fbd0, 0x9abc8bd5, 0x9e7d9662,
9780 0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
9781 0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
9782 };

9783 unsigned long memcrc(const unsigned char *b, size_t n)
9784 {
9785 /* Input arguments:
9786 * const char* b == byte sequence to checksum
9787 * size_t n == length of sequence
9788 */

9789 register unsigned i, c, s = 0;

9790 for (i = n; i > 0; --i) {
9791 c = (unsigned) (*b++);
9792 s = (s << 8) ^ crctab[(s >> 24) ^ c];
9793 }

9794 /* Extend with the length of the string. */
9795 while (n != 0) {
9796 c = n & 0377;
9797 n >>= 8;
9798 s = (s << 8) ^ crctab[(s >> 24) ^ c];
9799 }

9800 return ~s;
9801 }

```

The historical practice of writing the number of “blocks” has been changed to writing the number of octets, since the latter is not only more useful, but also since historical implementations have not been consistent in defining what a “block” meant. Octets are used instead of bytes because bytes can differ in size between systems.

The algorithm used was selected to increase the operational robustness of *cksum*. Neither the System V nor BSD *sum* algorithm was selected. Since each of these was different and each was the default behavior on those systems, no realistic compromise was available if either were selected—some set of historical applications would break. Therefore, the name was changed to *cksum*. Although the historical *sum* commands will probably continue to be provided for many years, programs designed for portability across systems should use the new name.

The algorithm selected is based on that used by the ISO/IEC 8802-3:1996 standard (Ethernet) for the frame check sequence field. The algorithm used does not match the technical definition of a *checksum*; the term is used for historical reasons. The length of the file is included in the CRC calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also because it guards against inadvertent collisions between files that begin with different series of zero octets. The chance that two different files produce identical CRCs is much greater when their lengths are not considered. Keeping the length and the checksum of the file itself separate would yield a slightly more robust algorithm, but historical usage has always been that a single number (the checksum as printed) represents the signature of the file. It was decided that

9821 historical usage was the more important consideration.

9822 Early proposals contained modifications to the Ethernet algorithm that involved extracting table  
9823 values whenever an intermediate result became zero. This was demonstrated to be less robust  
9824 than the current method and mathematically difficult to describe or justify.

9825 The calculation used is identical to that given in pseudo-code in the referenced Sarwate article.  
9826 The pseudo-code rendition is:

```
9827 X ← 0; Y ← 0;
9828 for i ← m -1 step -1 until 0 do
9829 begin
9830 T ← X(1) ^ A[i];
9831 X(1) ← X(0); X(0) ← Y(1); Y(1) ← Y(0); Y(0) ← 0;
9832 comment: f[T] and f'[T] denote the T-th words in the
9833 table f and f' ;
9834 X ← X ^ f[T]; Y ← Y ^ f'[T];
9835 end
```

9836 The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]**  
9837 represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables  
9838 **f** and **f'** are a single table containing 32-bit values.

9839 The referenced Sarwate article also discusses generating the table.

9840 **FUTURE DIRECTIONS**

9841 None.

9842 **SEE ALSO**

9843 None.

9844 **CHANGE HISTORY**

9845 First released in Issue 4.



9846 **NAME**

9847 cmp — compare two files

9848 **SYNOPSIS**9849 cmp [ -l | -s ] *file1 file2*9850 **DESCRIPTION**

9851 The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the  
 9852 same. Under default options, if they differ, it shall write to standard output the byte and line  
 9853 number at which the first difference occurred. Bytes and lines shall be numbered beginning with  
 9854 1.

9855 **OPTIONS**

9856 The *cmp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 9857 12.2, Utility Syntax Guidelines.

9858 The following options shall be supported:

9859 **-l** (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for  
 9860 each difference.

9861 **-s** Write nothing for differing files; return exit status only.

9862 **OPERANDS**

9863 The following operands shall be supported:

9864 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall  
 9865 be used.

9866 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input  
 9867 shall be used.

9868 If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or  
 9869 character special file, the results are undefined.

9870 **STDIN**

9871 The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the  
 9872 INPUT FILES section.

9873 **INPUT FILES**

9874 The input files can be any file type.

9875 **ENVIRONMENT VARIABLES**

9876 The following environment variables shall affect the execution of *cmp*:

9877 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 9878 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 9879 Internationalization Variables for the precedence of internationalization variables  
 9880 used to determine the values of locale categories.)

9881 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 9882 internationalization variables.

9883 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 9884 characters (for example, single-byte as opposed to multi-byte characters in  
 9885 arguments).

9886 **LC\_MESSAGES**

9887 Determine the locale that should be used to affect the format and contents of  
 9888 diagnostic messages written to standard error and informative messages written to  
 9889 standard output.

9890 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 9891 ASYNCHRONOUS EVENTS

9892 Default.

## 9893 STDOUT

9894 In the POSIX locale, results of the comparison shall be written to standard output. When no  
9895 options are used, the format shall be:

9896 "%s %s differ: char %d, line %d\n", *file1*, *file2*,  
9897 <byte number>, <line number>

9898 When the **-I** option is used, the format shall be:

9899 "%d %o %o\n", <byte number>, <differing byte>,  
9900 <differing byte>

9901 for each byte that differs. The first <differing byte> number is from *file1* while the second is from  
9902 *file2*. In both cases, <byte number> shall be relative to the beginning of the file, beginning with 1.

9903 No output shall be written to standard output when the **-s** option is used.

## 9904 STDERR

9905 The standard error shall be used only for diagnostic messages. If *file1* and *file2* are identical for  
9906 the entire length of the shorter file, in the POSIX locale the following diagnostic message shall be  
9907 written, unless the **-s** option is specified:

9908 "cmp: EOF on %s%s\n", <name of shorter file>, <additional info>

9909 The <additional info> field shall either be null or a string that starts with a <blank> and contains  
9910 no <newline>s. Some implementations report on the number of lines in this case.

## 9911 OUTPUT FILES

9912 None.

## 9913 EXTENDED DESCRIPTION

9914 None.

## 9915 EXIT STATUS

9916 The following exit values shall be returned:

9917 0 The files are identical.

9918 1 The files are different; this includes the case where one file is identical to the first part of the  
9919 other.

9920 >1 An error occurred.

## 9921 CONSEQUENCES OF ERRORS

9922 Default.

## 9923 APPLICATION USAGE

9924 Although input files to *cmp* can be any type, the results might not be what would be expected on  
9925 character special device files or on file types not described by the System Interfaces volume of  
9926 IEEE Std 1003.1-2001. Since this volume of IEEE Std 1003.1-2001 does not specify the block size  
9927 used when doing input, comparisons of character special files need not compare all of the data  
9928 in those files.

9929 For files which are not text files, line numbers simply reflect the presence of a <newline>,  
9930 without any implication that the file is organized into lines.

9931 **EXAMPLES**

9932       None.

9933 **RATIONALE**

9934       The global language in Section 1.11 (on page 20) indicates that using two mutually-exclusive  
9935       options together produces unspecified results. Some System V implementations consider the  
9936       option usage:

9937       `cmp -l -s ...`

9938       to be an error. They also treat:

9939       `cmp -s -l ...`

9940       as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

9941       The word **char** in the standard output format comes from historical usage, even though it is  
9942       actually a byte number. When *cmp* is supported in other locales, implementations are  
9943       encouraged to use the word *byte* or its equivalent in another language. Users should not  
9944       interpret this difference to indicate that the functionality of the utility changed between locales.

9945       Some implementations report on the number of lines in the identical-but-shorter file case. This is  
9946       allowed by the inclusion of the *<additional info>* fields in the output format. The restriction on  
9947       having a leading *<blank>* and no *<newline>*s is to make parsing for the filename easier. It is  
9948       recognized that some filenames containing white-space characters make parsing difficult  
9949       anyway, but the restriction does aid programs used on systems where the names are  
9950       predominantly well behaved.

9951 **FUTURE DIRECTIONS**

9952       None.

9953 **SEE ALSO**9954       *comm*, *diff*9955 **CHANGE HISTORY**

9956       First released in Issue 2.

9957 **NAME**

9958           comm — select or reject lines common to two files

9959 **SYNOPSIS**9960           comm [-123] *file1 file2*9961 **DESCRIPTION**

9962           The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and lines in both files.

9965           If the lines in both files are not ordered according to the collating sequence of the current locale, the results are unspecified.

9967 **OPTIONS**

9968           The *comm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines.

9970           The following options shall be supported:

9971           -1           Suppress the output column of lines unique to *file1*.

9972           -2           Suppress the output column of lines unique to *file2*.

9973           -3           Suppress the output column of lines duplicated in *file1* and *file2*.

9974 **OPERANDS**

9975           The following operands shall be supported:

9976           *file1*       A pathname of the first file to be compared. If *file1* is '-', the standard input shall be used.

9978           *file2*       A pathname of the second file to be compared. If *file2* is '-', the standard input shall be used.

9980           If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or character special file, the results are undefined.

9982 **STDIN**

9983           The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input. See the INPUT FILES section.

9985 **INPUT FILES**

9986           The input files shall be text files.

9987 **ENVIRONMENT VARIABLES**

9988           The following environment variables shall affect the execution of *comm*:

9989           *LANG*       Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

9993           *LC\_ALL*     If set to a non-empty string value, override the values of all the other internationalization variables.

9995           *LC\_COLLATE* Determine the locale for the collating sequence *comm* expects to have been used when the input files were sorted.

9998           *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in

10000 arguments and input files).

10001 **LC\_MESSAGES**

10002 Determine the locale that should be used to affect the format and contents of

10003 diagnostic messages written to standard error.

10004 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

10005 **ASYNCHRONOUS EVENTS**

10006 Default.

10007 **STDOUT**

10008 The *comm* utility shall produce output depending on the options selected. If the **-1**, **-2**, and **-3**

10009 options are all selected, *comm* shall write nothing to standard output.

10010 If the **-1** option is not selected, lines contained only in *file1* shall be written using the format:

10011 "%s\n", <line in file1>

10012 If the **-2** option is not selected, lines contained only in *file2* are written using the format:

10013 "%s%s\n", <lead>, <line in file2>

10014 where the string <lead> is as follows:

10015 <tab> The **-1** option is not selected.

10016 null string The **-1** option is selected.

10017 If the **-3** option is not selected, lines contained in both files shall be written using the format:

10018 "%s%s\n", <lead>, <line in both>

10019 where the string <lead> is as follows:

10020 <tab><tab> Neither the **-1** nor the **-2** option is selected.

10021 <tab> Exactly one of the **-1** and **-2** options is selected.

10022 null string Both the **-1** and **-2** options are selected.

10023 If the input files were ordered according to the collating sequence of the current locale, the lines

10024 written shall be in the collating sequence of the original lines.

10025 **STDERR**

10026 The standard error shall be used only for diagnostic messages.

10027 **OUTPUT FILES**

10028 None.

10029 **EXTENDED DESCRIPTION**

10030 None.

10031 **EXIT STATUS**

10032 The following exit values shall be returned:

10033 0 All input files were successfully output as specified.

10034 >0 An error occurred.

10035 **CONSEQUENCES OF ERRORS**

10036 Default.

**10037 APPLICATION USAGE**

10038 If the input files are not properly presorted, the output of *comm* might not be useful.

**10039 EXAMPLES**

10040 If a file named **xcu** contains a sorted list of the utilities in this volume of IEEE Std 1003.1-2001, a  
10041 file named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide,  
10042 Issue 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface  
10043 Definition Third Edition:

10044 `comm -23 xcu xpg3 | comm -23 - svid89`

10045 would print a list of utilities in this volume of IEEE Std 1003.1-2001 not specified by either of the  
10046 other documents:

10047 `comm -12 xcu xpg3 | comm -12 - svid89`

10048 would print a list of utilities specified by all three documents, and:

10049 `comm -12 xpg3 svid89 | comm -23 - xcu`

10050 would print a list of utilities specified by both XPG3 and the SVID, but not specified in this  
10051 volume of IEEE Std 1003.1-2001.

**10052 RATIONALE**

10053 None.

**10054 FUTURE DIRECTIONS**

10055 None.

**10056 SEE ALSO**

10057 *cmp, diff, sort, uniq*

**10058 CHANGE HISTORY**

10059 First released in Issue 2.

**10060 Issue 6**

10061 The normative text is reworded to avoid use of the term “must” for application requirements.

10062 **NAME**10063        `command` — execute a simple command10064 **SYNOPSIS**10065        `command` [`-p`] *command\_name* [*argument* ...]10066 UP     `command` [`-v` | `-V`] *command\_name*

10067

10068 **DESCRIPTION**10069        The *command* utility shall cause the shell to treat the arguments as a simple command,  
10070        suppressing the shell function lookup that is described in Section 2.9.1.1 (on page 48), item 1b.10071        If the *command\_name* is the same as the name of one of the special built-in utilities, the special  
10072        properties in the enumerated list at the beginning of Section 2.14 (on page 64) shall not occur. In  
10073        every other respect, if *command\_name* is not the name of a function, the effect of *command* (with  
10074        no options) shall be the same as omitting *command*.10075        On systems supporting the User Portability Utilities option, the *command* utility also shall  
10076        provide information concerning how a command name is interpreted by the shell; see `-v` and  
10077        `-V`.10078 **OPTIONS**10079        The *command* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001,  
10080        Section 12.2, Utility Syntax Guidelines.

10081        The following options shall be supported:

10082        `-p`        Perform the command search using a default value for *PATH* that is guaranteed to  
10083        find all of the standard utilities.10084        `-v`        (On systems supporting the User Portability Utilities option.) Write a string to  
10085        standard output that indicates the pathname or command that will be used by the  
10086        shell, in the current shell execution environment (see Section 2.12 (on page 61)), to  
10087        invoke *command\_name*, but do not invoke *command\_name*.10088                • Utilities, regular built-in utilities, *command\_names* including a slash character,  
10089                and any implementation-defined functions that are found using the *PATH*  
10090                variable (as described in Section 2.9.1.1 (on page 48)), shall be written as  
10091                absolute pathnames.10092                • Shell functions, special built-in utilities, regular built-in utilities not associated  
10093                with a *PATH* search, and shell reserved words shall be written as just their  
10094                names.

10095                • An alias shall be written as a command line that represents its alias definition.

10096                • Otherwise, no output shall be written and the exit status shall reflect that the  
10097                name was not found.10098        `-V`        (On systems supporting the User Portability Utilities option.) Write a string to  
10099        standard output that indicates how the name given in the *command\_name* operand  
10100        will be interpreted by the shell, in the current shell execution environment (see  
10101        Section 2.12 (on page 61)), but do not invoke *command\_name*. Although the format  
10102        of this string is unspecified, it shall indicate in which of the following categories  
10103        *command\_name* falls and shall include the information stated:10104                • Utilities, regular built-in utilities, and any implementation-defined functions  
10105                that are found using the *PATH* variable (as described in Section 2.9.1.1 (on page  
10106                48)), shall be identified as such and include the absolute pathname in the string.

- 10107 • Other shell functions shall be identified as functions.
- 10108 • Aliases shall be identified as aliases and their definitions included in the string.
- 10109 • Special built-in utilities shall be identified as special built-in utilities.
- 10110 • Regular built-in utilities not associated with a *PATH* search shall be identified
- 10111 as regular built-in utilities. (The term “regular” need not be used.)
- 10112 • Shell reserved words shall be identified as reserved words.

**10113 OPERANDS**

10114 The following operands shall be supported:

10115 *argument* One of the strings treated as an argument to *command\_name*.

10116 *command\_name*

10117 The name of a utility or a special built-in utility.

**10118 STDIN**

10119 Not used.

**10120 INPUT FILES**

10121 None.

**10122 ENVIRONMENT VARIABLES**

10123 The following environment variables shall affect the execution of *command*:

10124 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 10125 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 10126 Internationalization Variables for the precedence of internationalization variables  
 10127 used to determine the values of locale categories.)

10128 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 10129 internationalization variables.

10130 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 10131 characters (for example, single-byte as opposed to multi-byte characters in  
 10132 arguments).

10133 *LC\_MESSAGES*

10134 Determine the locale that should be used to affect the format and contents of  
 10135 diagnostic messages written to standard error and informative messages written to  
 10136 standard output.

10137 *XS1* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

10138 *PATH* Determine the search path used during the command search described in Section  
 10139 2.9.1.1 (on page 48), except as described under the **-p** option.

**10140 ASYNCHRONOUS EVENTS**

10141 Default.

**10142 STDOUT**

10143 When the **-v** option is specified, standard output shall be formatted as:

10144 "%s\n", *<pathname or command>*

10145 When the **-V** option is specified, standard output shall be formatted as:

10146 "%s\n", *<unspecified>*



10147 **STDERR**

10148       The standard error shall be used only for diagnostic messages.

10149 **OUTPUT FILES**

10150       None.

10151 **EXTENDED DESCRIPTION**

10152       None.

10153 **EXIT STATUS**

10154       When the **-v** or **-V** options are specified, the following exit values shall be returned:

10155       0   Successful completion.

10156       >0   The *command\_name* could not be found or an error occurred.

10157       Otherwise, the following exit values shall be returned:

10158       126   The utility specified by *command\_name* was found but could not be invoked.

10159       127   An error occurred in the *command* utility or the utility specified by *command\_name* could not  
10160       be found.

10161       Otherwise, the exit status of *command* shall be that of the simple command specified by the  
10162       arguments to *command*.

10163 **CONSEQUENCES OF ERRORS**

10164       Default.

10165 **APPLICATION USAGE**

10166       The order for command search allows functions to override regular built-ins and path searches.  
10167       This utility is necessary to allow functions that have the same name as a utility to call the utility  
10168       (instead of a recursive call to the function).

10169       The system default path is available using *getconf*; however, since *getconf* may need to have the  
10170       *PATH* set up before it can be called itself, the following can be used:

10171       `command -p getconf _CS_PATH`

10172       There are some advantages to suppressing the special characteristics of special built-ins on  
10173       occasion. For example:

10174       `command exec > unwritable-file`

10175       does not cause a non-interactive script to abort, so that the output status can be checked by the  
10176       script.

10177       The *command*, *env*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an  
10178       error occurs so that applications can distinguish “failure to find a utility” from “invoked utility  
10179       exited with an error indication”. The value 127 was chosen because it is not commonly used for  
10180       other meanings; most utilities use small values for “normal error conditions” and the values  
10181       above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen  
10182       in a similar manner to indicate that the utility could be found, but not invoked. Some scripts  
10183       produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
10184       between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
10185       *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
10186       any other reason.

10187       Since the **-v** and **-V** options of *command* produce output in relation to the current shell execution  
10188       environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell  
10189       or separate utility execution environment, such as one of the following:

```

10190 (PATH=foo command -v)
10191 nohup command -v

```

10192 it does not necessarily produce correct results. For example, when called with *nohup* or an *exec*  
 10193 function, in a separate utility execution environment, most implementations are not able to  
 10194 identify aliases, functions, or special built-ins.

10195 Two types of regular built-ins could be encountered on a system and these are described  
 10196 separately by *command*. The description of command search in Section 2.9.1.1 (on page 48)  
 10197 allows for a standard utility to be implemented as a regular built-in as long as it is found in the  
 10198 appropriate place in a *PATH* search. So, for example, *command -v true* might yield **/bin/true** or  
 10199 some similar pathname. Other implementation-defined utilities that are not defined by this  
 10200 volume of IEEE Std 1003.1-2001 might exist only as built-ins and have no pathname associated  
 10201 with them. These produce output identified as (regular) built-ins. Applications encountering  
 10202 these are not able to count on *execing* them, using them with *nohup*, overriding them with a  
 10203 different *PATH*, and so on.

#### 10204 EXAMPLES

- 10205 1. Make a version of *cd* that always prints out the new working directory exactly once:

```

10206 cd() {
10207 command cd "$@" >/dev/null
10208 pwd
10209 }

```

- 10210 2. Start off a “secure shell script” in which the script avoids being spoofed by its parent:

```

10211 IFS='
10212 '
10213 # The preceding value should be <space><tab><newline>.
10214 # Set IFS to its default value.

10215 \unalias -a
10216 # Unset all possible aliases.
10217 # Note that unalias is escaped to prevent an alias
10218 # being used for unalias.

10219 unset -f command
10220 # Ensure command is not a user function.

10221 PATH="$ (command -p getconf _CS_PATH) :$PATH"
10222 # Put on a reliable PATH prefix.

10223 ...

```

10224 At this point, given correct permissions on the directories called by *PATH*, the script has  
 10225 the ability to ensure that any utility it calls is the intended one. It is being very cautious  
 10226 because it assumes that implementation extensions may be present that would allow user  
 10227 functions to exist when it is invoked; this capability is not specified by this volume of  
 10228 IEEE Std 1003.1-2001, but it is not prohibited as an extension. For example, the *ENV*  
 10229 variable precedes the invocation of the script with a user start-up script. Such a script  
 10230 could define functions to spoof the application.

#### 10231 RATIONALE

10232 Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

10233 There is nothing in the description of *command* that implies the command line is parsed any  
 10234 differently from that of any other simple command. For example:

10235 `command a | b ; c`

10236 is not parsed in any special way that causes ' | ' or ' ; ' to be treated other than a pipe operator  
10237 or semicolon or that prevents function lookup on **b** or **c**.

10238 The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since  
10239 *command* also goes to the file system to search for utilities, the name *builtin* would not be  
10240 intuitive.

10241 The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special  
10242 built-in for the following reasons:

- 10243 • The removal of exportable functions made the special precedence of a special built-in  
10244 unnecessary.
- 10245 • A special built-in has special properties (see Section 2.14 (on page 64)) that were  
10246 inappropriate for invoking other utilities. For example, two commands such as:

10247 `date > unwritable-file`

10248 `command date > unwritable-file`

10249 would have entirely different results; in a non-interactive script, the former would continue  
10250 to execute the next command, the latter would abort. Introducing this semantic difference  
10251 along with suppressing functions was seen to be non-intuitive.

10252 The **-p** option is present because it is useful to be able to ensure a safe path search that finds all  
10253 the standard utilities. This search might not be identical to the one that occurs through one of the  
10254 *exec* functions (as defined in the System Interfaces volume of IEEE Std 1003.1-2001) when *PATH*  
10255 is unset. At the very least, this feature is required to allow the script to access the correct version  
10256 of *getconf* so that the value of the default path can be accurately retrieved.

10257 The *command* **-v** and **-V** options were added to satisfy requirements from users that are  
10258 currently accomplished by three different historical utilities: *type* in the System V shell, *whence* in  
10259 the KornShell, and *which* in the C shell. Since there is no historical agreement on how and what  
10260 to accomplish here, the POSIX *command* utility was enhanced and the historical utilities were left  
10261 unmodified. The C shell *which* merely conducts a path search. The KornShell *whence* is more  
10262 elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases,  
10263 exported aliases, and undefined functions.

10264 The output format of **-V** was left mostly unspecified because human users are its only audience.  
10265 Applications should not be written to care about this information; they can use the output of **-v**  
10266 to differentiate between various types of commands, but the additional information that may be  
10267 emitted by the more verbose **-V** is not needed and should not be arbitrarily constrained in its  
10268 verbosity or localization for application parsing reasons.

#### 10269 **FUTURE DIRECTIONS**

10270 None.

#### 10271 **SEE ALSO**

10272 Section 2.9.1.1 (on page 48), Section 2.12 (on page 61), Section 2.14 (on page 64), *sh*, *type*, the  
10273 System Interfaces volume of IEEE Std 1003.1-2001, *exec*

#### 10274 **CHANGE HISTORY**

10275 First released in Issue 4.

10276 **NAME**

10277           compress — compress data

10278 **SYNOPSIS**10279 xSI       compress [-fv] [-b *bits*] [*file* ...]10280           compress [-cfv] [-b *bits*] [*file*]

10281

10282 **DESCRIPTION**

10283       The *compress* utility shall attempt to reduce the size of the named files by using adaptive  
 10284       Lempel-Ziv coding algorithm.

10285       **Note:**       Lempel-Ziv is US Patent 4464650, issued to William Eastman, Abraham Lempel, Jacob Ziv,  
 10286       Martin Cohn on August 7th, 1984, and assigned to Sperry Corporation.

10287       Lempel-Ziv-Welch compression is covered by US Patent 4558302, issued to Terry A. Welch on  
 10288       December 10th, 1985, and assigned to Sperry Corporation.

10289       On systems not supporting adaptive Lempel-Ziv coding algorithm, the input files shall not be  
 10290       changed and an error value greater than two shall be returned. Except when the output is to the  
 10291       standard output, each file shall be replaced by one with the extension **.Z**. If the invoking process  
 10292       has appropriate privileges, the ownership, modes, access time, and modification time of the  
 10293       original file are preserved. If appending the **.Z** to the filename would make the name exceed  
 10294       {NAME\_MAX} bytes, the command shall fail. If no files are specified, the standard input shall be  
 10295       compressed to the standard output.

10296 **OPTIONS**

10297       The *compress* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001,  
 10298       Section 12.2, Utility Syntax Guidelines.

10299       The following options shall be supported:

10300       **-b bits**       Specify the maximum number of bits to use in a code. For a conforming  
 10301       application, the *bits* argument shall be:

10302               9 <= *bits* <= 14

10303       The implementation may allow *bits* values of greater than 14. The default is 14, 15,  
 10304       or 16.

10305       **-c**           Cause *compress* to write to the standard output; the input file is not changed, and  
 10306       no **.Z** files are created.

10307       **-f**           Force compression of *file*, even if it does not actually reduce the size of the file, or if  
 10308       the corresponding *file.Z* file already exists. If the **-f** option is not given, and the  
 10309       process is not running in the background, the user is prompted as to whether an  
 10310       existing *file.Z* file should be overwritten.

10311       **-v**           Write the percentage reduction of each file to standard error.

10312 **OPERANDS**

10313       The following operand shall be supported:

10314       *file*        A pathname of a file to be compressed.

10315 **STDIN**

10316       The standard input shall be used only if no *file* operands are specified, or if a *file* operand is **'-'**.

10317 **INPUT FILES**

10318       If *file* operands are specified, the input files contain the data to be compressed.

10319 **ENVIRONMENT VARIABLES**

10320       The following environment variables shall affect the execution of *compress*:

10321       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 10322                   (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 10323                   Internationalization Variables for the precedence of internationalization variables  
 10324                   used to determine the values of locale categories.)

10325       *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 10326                   internationalization variables.

10327       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 10328                   characters (for example, single-byte as opposed to multi-byte characters in  
 10329                   arguments).

10330       *LC\_MESSAGES*  
 10331                   Determine the locale that should be used to affect the format and contents of  
 10332                   diagnostic messages written to standard error.

10333       *NLSPATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

10334 **ASYNCHRONOUS EVENTS**

10335       Default.

10336 **STDOUT**

10337       If no *file* operands are specified, or if a *file* operand is '-', or if the *-c* option is specified, the  
 10338       standard output contains the compressed output.

10339 **STDERR**

10340       The standard error shall be used only for diagnostic and prompt messages and the output from  
 10341       *-v*.

10342 **OUTPUT FILES**

10343       The output files shall contain the compressed output. The format of compressed files is  
 10344       unspecified and interchange of such files between implementations (including access via  
 10345       unspecified file sharing mechanisms) is not required by IEEE Std 1003.1-2001.

10346 **EXTENDED DESCRIPTION**

10347       None.

10348 **EXIT STATUS**

10349       The following exit values shall be returned:

- 10350       0   Successful completion.
- 10351       1   An error occurred.
- 10352       2   One or more files were not compressed because they would have increased in size (and the  
 10353       *-f* option was not specified).
- 10354       >2   An error occurred.

10355 **CONSEQUENCES OF ERRORS**

10356       The input file shall remain unmodified.

10357 **APPLICATION USAGE**

10358       The amount of compression obtained depends on the size of the input, the number of *bits* per  
10359       code, and the distribution of common substrings. Typically, text such as source code or English  
10360       is reduced by 50-60%. Compression is generally much better than that achieved by Huffman  
10361       coding or adaptive Huffman coding (*compact*), and takes less time to compute.

10362       Although *compress* strictly follows the default actions upon receipt of a signal or when an error  
10363       occurs, some unexpected results may occur. In some implementations it is likely that a partially  
10364       compressed file is left in place, alongside its uncompressed input file. Since the general  
10365       operation of *compress* is to delete the uncompressed file only after the *.Z* file has been  
10366       successfully filled, an application should always carefully check the exit status of *compress* before  
10367       arbitrarily deleting files that have like-named neighbors with *.Z* suffixes.

10368       The limit of 14 on the *bits* option-argument is to achieve portability to all systems (within the  
10369       restrictions imposed by the lack of an explicit published file format). Some implementations  
10370       based on 16-bit architectures cannot support 15 or 16-bit uncompression.

10371 **EXAMPLES**

10372       None.

10373 **RATIONALE**

10374       None.

10375 **FUTURE DIRECTIONS**

10376       None.

10377 **SEE ALSO**

10378       *uncompress*, *zcat*

10379 **CHANGE HISTORY**

10380       First released in Issue 4.

10381 **Issue 6**

10382       The normative text is reworded to avoid use of the term “must” for application requirements.

10383       An error case is added for systems not supporting adaptive Lempel-Ziv coding.

10384 **NAME**

10385 cp — copy files

10386 **SYNOPSIS**10387 cp [-fip] *source\_file target\_file*10388 cp [-fip] *source\_file ... target*10389 cp -R [-H | -L | -P] [-fip] *source\_file ... target*10390 OB cp -r [-H | -L | -P] [-fip] *source\_file ... target*10391 **DESCRIPTION**

10392 The first synopsis form is denoted by two operands, neither of which are existing files of type  
 10393 directory. The *cp* utility shall copy the contents of *source\_file* (or, if *source\_file* is a file of type  
 10394 symbolic link, the contents of the file referenced by *source\_file*) to the destination path named by  
 10395 *target\_file*.

10396 The second synopsis form is denoted by two or more operands where the **-R** or **-r** options are  
 10397 not specified and the first synopsis form is not applicable. It shall be an error if any *source\_file* is a  
 10398 file of type directory, if *target* does not exist, or if *target* is a file of a type defined by the System  
 10399 Interfaces volume of IEEE Std 1003.1-2001, but is not a file of type directory. The *cp* utility shall  
 10400 copy the contents of each *source\_file* (or, if *source\_file* is a file of type symbolic link, the contents  
 10401 of the file referenced by *source\_file*) to the destination path named by the concatenation of *target*,  
 10402 a slash character, and the last component of *source\_file*.

10403 The third and fourth synopsis forms are denoted by two or more operands where the **-R** or **-r**  
 10404 options are specified. The *cp* utility shall copy each file in the file hierarchy rooted in each  
 10405 *source\_file* to a destination path named as follows:

- 10406 • If *target* exists and is a file of type directory, the name of the corresponding destination path  
 10407 for each file in the file hierarchy shall be the concatenation of *target*, a slash character, and the  
 10408 pathname of the file relative to the directory containing *source\_file*.
- 10409 • If *target* does not exist and two operands are specified, the name of the corresponding  
 10410 destination path for *source\_file* shall be *target*; the name of the corresponding destination path  
 10411 for all other files in the file hierarchy shall be the concatenation of *target*, a slash character,  
 10412 and the pathname of the file relative to *source\_file*.

10413 It shall be an error if *target* does not exist and more than two operands are specified, or if *target*  
 10414 exists and is a file of a type defined by the System Interfaces volume of IEEE Std 1003.1-2001, but  
 10415 is not a file of type directory.

10416 In the following description, the term *dest\_file* refers to the file named by the destination path.  
 10417 The term *source\_file* refers to the file that is being copied, whether specified as an operand or a  
 10418 file in a file hierarchy rooted in a *source\_file* operand. If *source\_file* is a file of type symbolic link:

- 10419 • If neither the **-R** nor **-r** options were specified, *cp* shall take actions based on the type and  
 10420 contents of the file referenced by the symbolic link, and not by the symbolic link itself.
- 10421 • If the **-R** option was specified:
  - 10422 — If none of the options **-H**, **-L**, nor **-P** were specified, it is unspecified which of **-H**, **-L**, or  
 10423 **-P** will be used as a default.
  - 10424 — If the **-H** option was specified, *cp* shall take actions based on the type and contents of the  
 10425 file referenced by any symbolic link specified as a *source\_file* operand.
  - 10426 — If the **-L** option was specified, *cp* shall take actions based on the type and contents of the  
 10427 file referenced by any symbolic link specified as a *source\_file* operand or any symbolic

10428 links encountered during traversal of a file hierarchy.

10429 — If the **-P** option was specified, *cp* shall copy any symbolic link specified as a *source\_file*  
 10430 operand and any symbolic links encountered during traversal of a file hierarchy, and shall  
 10431 not follow any symbolic links.

10432 • If the **-r** option was specified, the behavior is implementation-defined.

10433 For each *source\_file*, the following steps shall be taken:

10434 1. If *source\_file* references the same file as *dest\_file*, *cp* may write a diagnostic message to  
 10435 standard error; it shall do nothing more with *source\_file* and shall go on to any remaining  
 10436 files.

10437 2. If *source\_file* is of type directory, the following steps shall be taken:

10438 a. If neither the **-R** or **-r** options were specified, *cp* shall write a diagnostic message to  
 10439 standard error, do nothing more with *source\_file*, and go on to any remaining files.

10440 b. If *source\_file* was not specified as an operand and *source\_file* is dot or dot-dot, *cp* shall  
 10441 do nothing more with *source\_file* and go on to any remaining files.

10442 c. If *dest\_file* exists and it is a file type not specified by the System Interfaces volume of  
 10443 IEEE Std 1003.1-2001, the behavior is implementation-defined.

10444 d. If *dest\_file* exists and it is not of type directory, *cp* shall write a diagnostic message to  
 10445 standard error, do nothing more with *source\_file* or any files below *source\_file* in the  
 10446 file hierarchy, and go on to any remaining files.

10447 e. If the directory *dest\_file* does not exist, it shall be created with file permission bits set  
 10448 to the same value as those of *source\_file*, modified by the file creation mask of the  
 10449 user if the **-p** option was not specified, and then bitwise-inclusively OR'ed with  
 10450 S\_IRWXU. If *dest\_file* cannot be created, *cp* shall write a diagnostic message to  
 10451 standard error, do nothing more with *source\_file*, and go on to any remaining files. It  
 10452 is unspecified if *cp* attempts to copy files in the file hierarchy rooted in *source\_file*.

10453 f. The files in the directory *source\_file* shall be copied to the directory *dest\_file*, taking  
 10454 the four steps (1 to 4) listed here with the files as *source\_files*.

10455 g. If *dest\_file* was created, its file permission bits shall be changed (if necessary) to be the  
 10456 same as those of *source\_file*, modified by the file creation mask of the user if the **-p**  
 10457 option was not specified.

10458 h. The *cp* utility shall do nothing more with *source\_file* and go on to any remaining files.

10459 3. If *source\_file* is of type regular file, the following steps shall be taken:

10460 a. If *dest\_file* exists, the following steps shall be taken:

10461 i. If the **-i** option is in effect, the *cp* utility shall write a prompt to the standard  
 10462 error and read a line from the standard input. If the response is not affirmative,  
 10463 *cp* shall do nothing more with *source\_file* and go on to any remaining files.

10464 ii. A file descriptor for *dest\_file* shall be obtained by performing actions equivalent  
 10465 to the *open()* function defined in the System Interfaces volume of  
 10466 IEEE Std 1003.1-2001 called using *dest\_file* as the *path* argument, and the  
 10467 bitwise-inclusive OR of O\_WRONLY and O\_TRUNC as the *oflag* argument.

10468 iii. If the attempt to obtain a file descriptor fails and the **-f** option is in effect, *cp*  
 10469 shall attempt to remove the file by performing actions equivalent to the  
 10470 *unlink()* function defined in the System Interfaces volume of



10471 IEEE Std 1003.1-2001 called using *dest\_file* as the *path* argument. If this attempt  
 10472 succeeds, *cp* shall continue with step 3b.

10473 b. If *dest\_file* does not exist, a file descriptor shall be obtained by performing actions  
 10474 equivalent to the *open()* function defined in the System Interfaces volume of  
 10475 IEEE Std 1003.1-2001 called using *dest\_file* as the *path* argument, and the bitwise-  
 10476 inclusive OR of *O\_WRONLY* and *O\_CREAT* as the *oflag* argument. The file  
 10477 permission bits of *source\_file* shall be the *mode* argument.

10478 c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to  
 10479 standard error, do nothing more with *source\_file*, and go on to any remaining files.

10480 d. The contents of *source\_file* shall be written to the file descriptor. Any write errors  
 10481 shall cause *cp* to write a diagnostic message to standard error and continue to step 3e.

10482 e. The file descriptor shall be closed.

10483 f. The *cp* utility shall do nothing more with *source\_file*. If a write error occurred in step  
 10484 3d, it is unspecified if *cp* continues with any remaining files. If no write error  
 10485 occurred in step 3d, *cp* shall go on to any remaining files.

10486 4. Otherwise, the following steps shall be taken:

10487 a. If the **-r** option was specified, the behavior is implementation-defined.

10488 b. If the **-R** option was specified, the following steps shall be taken:

10489 i. The *dest\_file* shall be created with the same file type as *source\_file*.

10490 ii. If *source\_file* is a file of type FIFO, the file permission bits shall be the same as  
 10491 those of *source\_file*, modified by the file creation mask of the user if the **-p**  
 10492 option was not specified. Otherwise, the permissions, owner ID, and group ID  
 10493 of *dest\_file* are implementation-defined.

10494 If this creation fails for any reason, *cp* shall write a diagnostic message to  
 10495 standard error, do nothing more with *source\_file*, and go on to any remaining  
 10496 files.

10497 iii. If *source\_file* is a file of type symbolic link, the pathname contained in *dest\_file*  
 10498 shall be the same as the pathname contained in *source\_file*.

10499 If this fails for any reason, *cp* shall write a diagnostic message to standard error,  
 10500 do nothing more with *source\_file*, and go on to any remaining files.

10501 If the implementation provides additional or alternate access control mechanisms (see the Base  
 10502 Definitions volume of IEEE Std 1003.1-2001, Section 4.4, File Access Permissions), their effect on  
 10503 copies of files is implementation-defined.

10504 **OPTIONS**

10505 The *cp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 10506 Utility Syntax Guidelines.

10507 The following options shall be supported:

10508 **-f** If a file descriptor for a destination file cannot be obtained, as described in step  
 10509 3.a.ii., attempt to unlink the destination file and proceed.

10510 **-H** Take actions based on the type and contents of the file referenced by any symbolic  
 10511 link specified as a *source\_file* operand.

10512 **-i** Write a prompt to standard error before copying to any existing destination file. If  
 10513 the response from the standard input is affirmative, the copy shall be attempted;

|       |                              |                                                                                                                                                                                                                                                                                                       |
|-------|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10514 |                              | otherwise, it shall not.                                                                                                                                                                                                                                                                              |
| 10515 | <b>-L</b>                    | Take actions based on the type and contents of the file referenced by any symbolic link specified as a <i>source_file</i> operand or any symbolic links encountered during traversal of a file hierarchy.                                                                                             |
| 10516 |                              |                                                                                                                                                                                                                                                                                                       |
| 10517 |                              |                                                                                                                                                                                                                                                                                                       |
| 10518 | <b>-P</b>                    | Take actions on any symbolic link specified as a <i>source_file</i> operand or any symbolic link encountered during traversal of a file hierarchy.                                                                                                                                                    |
| 10519 |                              |                                                                                                                                                                                                                                                                                                       |
| 10520 | <b>-p</b>                    | Duplicate the following characteristics of each source file in the corresponding destination file:                                                                                                                                                                                                    |
| 10521 |                              |                                                                                                                                                                                                                                                                                                       |
| 10522 |                              | 1. The time of last data modification and time of last access. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.                                                                                                                                |
| 10523 |                              |                                                                                                                                                                                                                                                                                                       |
| 10524 |                              | 2. The user ID and group ID. If this duplication fails for any reason, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.                                                                                                                                             |
| 10525 |                              |                                                                                                                                                                                                                                                                                                       |
| 10526 |                              | 3. The file permission bits and the S_ISUID and S_ISGID bits. Other, implementation-defined, bits may be duplicated as well. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.                                                                  |
| 10527 |                              |                                                                                                                                                                                                                                                                                                       |
| 10528 |                              |                                                                                                                                                                                                                                                                                                       |
| 10529 |                              | If the user ID or the group ID cannot be duplicated, the file permission bits S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but are not duplicated in the destination file, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.   |
| 10530 |                              |                                                                                                                                                                                                                                                                                                       |
| 10531 |                              |                                                                                                                                                                                                                                                                                                       |
| 10532 |                              |                                                                                                                                                                                                                                                                                                       |
| 10533 |                              | The order in which the preceding characteristics are duplicated is unspecified. The <i>dest_file</i> shall not be deleted if these characteristics cannot be preserved.                                                                                                                               |
| 10534 |                              |                                                                                                                                                                                                                                                                                                       |
| 10535 | <b>-R</b>                    | Copy file hierarchies.                                                                                                                                                                                                                                                                                |
| 10536 | OB <b>-r</b>                 | Copy file hierarchies. The treatment of special files is implementation-defined.                                                                                                                                                                                                                      |
| 10537 |                              | Specifying more than one of the mutually-exclusive options <b>-H</b> , <b>-L</b> , and <b>-P</b> shall not be considered an error. The last option specified shall determine the behavior of the utility.                                                                                             |
| 10538 |                              |                                                                                                                                                                                                                                                                                                       |
| 10539 | <b>OPERANDS</b>              |                                                                                                                                                                                                                                                                                                       |
| 10540 |                              | The following operands shall be supported:                                                                                                                                                                                                                                                            |
| 10541 | <i>source_file</i>           | A pathname of a file to be copied.                                                                                                                                                                                                                                                                    |
| 10542 | <i>target_file</i>           | A pathname of an existing or nonexistent file, used for the output when a single file is copied.                                                                                                                                                                                                      |
| 10543 |                              |                                                                                                                                                                                                                                                                                                       |
| 10544 | <i>target</i>                | A pathname of a directory to contain the copied files.                                                                                                                                                                                                                                                |
| 10545 | <b>STDIN</b>                 |                                                                                                                                                                                                                                                                                                       |
| 10546 |                              | The standard input shall be used to read an input line in response to each prompt specified in the STDERR section. Otherwise, the standard input shall not be used.                                                                                                                                   |
| 10547 |                              |                                                                                                                                                                                                                                                                                                       |
| 10548 | <b>INPUT FILES</b>           |                                                                                                                                                                                                                                                                                                       |
| 10549 |                              | The input files specified as operands may be of any file type.                                                                                                                                                                                                                                        |
| 10550 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                                                                                                                                                                       |
| 10551 |                              | The following environment variables shall affect the execution of <i>cp</i> :                                                                                                                                                                                                                         |
| 10552 | <i>LANG</i>                  | Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.) |
| 10553 |                              |                                                                                                                                                                                                                                                                                                       |
| 10554 |                              |                                                                                                                                                                                                                                                                                                       |
| 10555 |                              |                                                                                                                                                                                                                                                                                                       |

|       |                               |                                                                                                                                                                                                                                                                                                                                                          |
|-------|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10556 | <b>LC_ALL</b>                 | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                 |
| 10557 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10558 | <b>LC_COLLATE</b>             | Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the <b>yesexpr</b> locale keyword in the <b>LC_MESSAGES</b> category.                                                                                                                   |
| 10559 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10560 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10561 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10562 | <b>LC_CTYPE</b>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the <b>yesexpr</b> locale keyword in the <b>LC_MESSAGES</b> category. |
| 10563 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10564 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10565 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10566 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10567 | <b>LC_MESSAGES</b>            | Determine the locale for the processing of affirmative responses that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                 |
| 10568 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10569 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10570 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10571 | XSI <b>NLSPATH</b>            | Determine the location of message catalogs for the processing of <b>LC_MESSAGES</b> .                                                                                                                                                                                                                                                                    |
| 10572 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                          |
| 10573 |                               | Default.                                                                                                                                                                                                                                                                                                                                                 |
| 10574 | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                          |
| 10575 |                               | Not used.                                                                                                                                                                                                                                                                                                                                                |
| 10576 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                          |
| 10577 |                               | A prompt shall be written to standard error under the conditions specified in the <b>DESCRIPTION</b> section. The prompt shall contain the destination pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.                                                                      |
| 10578 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10579 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10580 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                          |
| 10581 |                               | The output files may be of any type.                                                                                                                                                                                                                                                                                                                     |
| 10582 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                          |
| 10583 |                               | None.                                                                                                                                                                                                                                                                                                                                                    |
| 10584 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                          |
| 10585 |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                             |
| 10586 | 0                             | All files were copied successfully.                                                                                                                                                                                                                                                                                                                      |
| 10587 | >0                            | An error occurred.                                                                                                                                                                                                                                                                                                                                       |
| 10588 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                          |
| 10589 |                               | If <b>cp</b> is prematurely terminated by a signal or error, files or file hierarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.                                                                                                                                           |
| 10590 |                               |                                                                                                                                                                                                                                                                                                                                                          |
| 10591 |                               |                                                                                                                                                                                                                                                                                                                                                          |

10592 **APPLICATION USAGE**

10593 The difference between **-R** and **-r** is in the treatment by *cp* of file types other than regular and  
 10594 directory. The original **-r** flag, for historic reasons, does not handle special files any differently  
 10595 from regular files, but always reads the file and copies its contents. This has obvious problems in  
 10596 the presence of special file types; for example, character devices, FIFOs, and sockets. The **-R**  
 10597 option is intended to recreate the file hierarchy and the **-r** option supports historical practice. It  
 10598 was anticipated that a future version of this volume of IEEE Std 1003.1-2001 would deprecate the  
 10599 **-r** option, and for that reason, there has been no attempt to fix its behavior with respect to FIFOs  
 10600 or other file types where copying the file is clearly wrong. However, some implementations  
 10601 support **-r** with the same abilities as the **-R** defined in this volume of IEEE Std 1003.1-2001. To  
 10602 accommodate them as well as systems that do not, the differences between **-r** and **-R** are  
 10603 implementation-defined. Implementations may make them identical. The **-r** option is marked  
 10604 obsolescent.

10605 The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to  
 10606 prevent users from creating programs that are set-user-ID or set-group-ID to them when  
 10607 copying files or to make set-user-ID or set-group-ID files accessible to new groups of users. For  
 10608 example, if a file is set-user-ID and the copy has a different group ID than the source, a new  
 10609 group of users has execute permission to a set-user-ID program than did previously. In  
 10610 particular, this is a problem for superusers copying users' trees.

10611 **EXAMPLES**

10612 None.

10613 **RATIONALE**

10614 The **-i** option exists on BSD systems, giving applications and users a way to avoid accidentally  
 10615 removing files when copying. Although the 4.3 BSD version does not prompt if the standard  
 10616 input is not a terminal, the standard developers decided that use of **-i** is a request for interaction,  
 10617 so when the destination path exists, the utility takes instructions from whatever responds on  
 10618 standard input.

10619 The exact format of the interactive prompts is unspecified. Only the general nature of the  
 10620 contents of prompts are specified because implementations may desire more descriptive  
 10621 prompts than those used on historical implementations. Therefore, an application using the **-i**  
 10622 option relies on the system to provide the most suitable dialog directly with the user, based on  
 10623 the behavior specified.

10624 The **-p** option is historical practice on BSD systems, duplicating the time of last data  
 10625 modification and time of last access. This volume of IEEE Std 1003.1-2001 extends it to preserve  
 10626 the user and group IDs, as well as the file permissions. This requirement has obvious problems  
 10627 in that the directories are almost certainly modified after being copied. This volume of  
 10628 IEEE Std 1003.1-2001 requires that the modification times be preserved. The statement that the  
 10629 order in which the characteristics are duplicated is unspecified is to permit implementations to  
 10630 provide the maximum amount of security for the user. Implementations should take into  
 10631 account the obvious security issues involved in setting the owner, group, and mode in the  
 10632 wrong order or creating files with an owner, group, or mode different from the final value.

10633 It is unspecified whether *cp* writes diagnostic messages when the user and group IDs cannot be  
 10634 set due to the widespread practice of users using **-p** to duplicate some portion of the file  
 10635 characteristics, indifferent to the duplication of others. Historic implementations only write  
 10636 diagnostic messages on errors other than [EPERM].

10637 The **-r** option is historical practice on BSD and BSD-derived systems, copying file hierarchies as  
 10638 opposed to single files. This functionality is used heavily in historical applications, and its loss  
 10639 would significantly decrease consensus. The **-R** option was added as a close synonym to the **-r**  
 10640 option, selected for consistency with all other options in this volume of IEEE Std 1003.1-2001 that

10641 do recursive directory descent.

10642 When a failure occurs during the copying of a file hierarchy, *cp* is required to attempt to copy  
10643 files that are on the same level in the hierarchy or above the file where the failure occurred. It is  
10644 unspecified if *cp* shall attempt to copy files below the file where the failure occurred (which  
10645 cannot succeed in any case).

10646 Permissions, owners, and groups of created special file types have been deliberately left as  
10647 implementation-defined. This is to allow systems to satisfy special requirements (for example,  
10648 allowing users to create character special devices, but requiring them to be owned by a certain  
10649 group). In general, it is strongly suggested that the permissions, owner, and group be the same  
10650 as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable  
10651 that additional privileges are required to create block, character, or other implementation-  
10652 defined special file types.

10653 Additionally, the **-p** option explicitly requires that all set-user-ID and set-group-ID permissions  
10654 be discarded if any of the owner or group IDs cannot be set. This is to keep users from  
10655 unintentionally giving away special privilege when copying programs.

10656 When creating regular files, historical versions of *cp* use the mode of the source file as modified  
10657 by the file mode creation mask. Other choices would have been to use the mode of the source file  
10658 unmodified by the creation mask or to use the same mode as would be given to a new file  
10659 created by the user (plus the execution bits of the source file) and then modify it by the file mode  
10660 creation mask. In the absence of any strong reason to change historic practice, it was in large part  
10661 retained.

10662 When creating directories, historical versions of *cp* use the mode of the source directory, plus  
10663 read, write, and search bits for the owner, as modified by the file mode creation mask. This is  
10664 done so that *cp* can copy trees where the user has read permission, but the owner does not. A  
10665 side effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the  
10666 copy is done, historical versions of *cp* set the permissions on the created directory to be the same  
10667 as the source directory, unmodified by the file creation mask.

10668 This behavior has been modified so that *cp* is always able to create the contents of the directory,  
10669 regardless of the file creation mask. After the copy is done, the permissions are set to be the same  
10670 as the source directory, as modified by the file creation mask. This latter change from historical  
10671 behavior is to prevent users from accidentally creating directories with permissions beyond  
10672 those they would normally set and for consistency with the behavior of *cp* in creating files.

10673 It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations  
10674 are strongly encouraged to do so. Historical implementations have detected the attempt in most  
10675 cases.

10676 There are two methods of copying subtrees in this volume of IEEE Std 1003.1-2001. The other  
10677 method is described as part of the *pax* utility (see *pax*). Both methods are historical practice. The  
10678 *cp* utility provides a simpler, more intuitive interface, while *pax* offers a finer granularity of  
10679 control. Each provides additional functionality to the other; in particular, *pax* maintains the  
10680 hard-link structure of the hierarchy, while *cp* does not. It is the intention of the standard  
10681 developers that the results be similar (using appropriate option combinations in both utilities).  
10682 The results are not required to be identical; there seemed insufficient gain to applications to  
10683 balance the difficulty of implementations having to guarantee that the results would be exactly  
10684 identical.

10685 The wording allowing *cp* to copy a directory to implementation-defined file types not specified  
10686 by the System Interfaces volume of IEEE Std 1003.1-2001 is provided so that implementations  
10687 supporting symbolic links are not required to prohibit copying directories to symbolic links.  
10688 Other extensions to the System Interfaces volume of IEEE Std 1003.1-2001 file types may need to

10689            use this loophole as well.

10690 **FUTURE DIRECTIONS**

10691            The **-r** option may be removed; use **-R** instead.

10692 **SEE ALSO**

10693            *mv*, *find*, *ln*, *pax*, the System Interfaces volume of IEEE Std 1003.1-2001, *open()*, *unlink()* |

10694 **CHANGE HISTORY**

10695            First released in Issue 2.

10696 **Issue 6**

10697            The **-r** option is marked obsolescent.

10698            The new options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
10699            options affect the processing of symbolic links.

10700            IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the **-P** option.

10701            IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/18 is applied, correcting an error in the |  
10702            SEE ALSO section. |

10703 **NAME**

10704 crontab — schedule periodic background work

10705 **SYNOPSIS**10706 UP crontab [*file*]10707 crontab [ *-e* | *-l* | *-r* ]

10708

10709 **DESCRIPTION**

10710 The *crontab* utility shall create, replace, or edit a user's crontab entry; a crontab entry is a list of  
 10711 commands and the times at which they shall be executed. The new crontab entry can be input by  
 10712 specifying *file* or input from standard input if no *file* operand is specified, or by using an editor, if  
 10713 *-e* is specified.

10714 Upon execution of a command from a crontab entry, the implementation shall supply a default  
 10715 environment, defining at least the following environment variables:

10716 *HOME* A pathname of the user's home directory.

10717 *LOGNAME* The user's login name.

10718 *PATH* A string representing a search path guaranteed to find all of the standard utilities.

10719 *SHELL* A pathname of the command interpreter. When *crontab* is invoked as specified by  
 10720 this volume of IEEE Std 1003.1-2001, the value shall be a pathname for *sh*.

10721 The values of these variables when *crontab* is invoked as specified by this volume of  
 10722 IEEE Std 1003.1-2001 shall not affect the default values provided when the scheduled command  
 10723 is run.

10724 If standard output and standard error are not redirected by commands executed from the  
 10725 crontab entry, any generated output or errors shall be mailed, via an implementation-defined  
 10726 method, to the user.

10727 XSI Users shall be permitted to use *crontab* if their names appear in the file */usr/lib/cron/cron.allow*.  
 10728 If that file does not exist, the file */usr/lib/cron/cron.deny* shall be checked to determine whether  
 10729 the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate  
 10730 privileges shall be allowed to submit a job. If only *cron.deny* exists and is empty, global usage  
 10731 shall be permitted. The *cron.allow* and *cron.deny* files shall consist of one user name per line.

10732 **OPTIONS**

10733 The *crontab* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 10734 12.2, Utility Syntax Guidelines.

10735 The following options shall be supported:

10736 *-e* Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if  
 10737 the crontab entry does not exist. When editing is complete, the entry shall be  
 10738 installed as the user's crontab entry.

10739 *-l* (The letter ell.) List the invoking user's crontab entry.

10740 *-r* Remove the invoking user's crontab entry.

10741 **OPERANDS**

10742 The following operand shall be supported:

10743 *file* The pathname of a file that contains specifications, in the format defined in the  
 10744 INPUT FILES section, for crontab entries.

10745 **STDIN**

10746 See the INPUT FILES section.

10747 **INPUT FILES**

10748 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file  
 10749 consisting of lines of six fields each. The fields shall be separated by <blank>s. The first five  
 10750 fields shall be integer patterns that specify the following:

- 10751 1. Minute [0,59]
- 10752 2. Hour [0,23]
- 10753 3. Day of the month [1,31]
- 10754 4. Month of the year [1,12]
- 10755 5. Day of the week ([0,6] with 0=Sunday)

10756 Each of these patterns can be either an asterisk (meaning all valid values), an element, or a list of  
 10757 elements separated by commas. An element shall be either a number or two numbers separated  
 10758 by a hyphen (meaning an inclusive range). The specification of days can be made by two fields  
 10759 (day of the month and day of the week). If month, day of month, and day of week are all  
 10760 asterisks, every day shall be matched. If either the month or day of month is specified as an  
 10761 element or list, but the day of week is an asterisk, the month and day of month fields shall  
 10762 specify the days that match. If both month and day of month are specified as an asterisk, but day  
 10763 of week is an element or list, then only the specified days of the week match. Finally, if either the  
 10764 month or day of month is specified as an element or list, and the day of week is also specified as  
 10765 an element or list, then any day matching either the month and day of month, or the day of  
 10766 week, shall be matched.

10767 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified  
 10768 times. A percent sign character in this field shall be translated to a <newline>. Any character  
 10769 preceded by a backslash (including the '%') shall cause that character to be treated literally.  
 10770 Only the first line (up to a '%' or end-of-line) of the command field shall be executed by the  
 10771 command interpreter. The other lines shall be made available to the command as standard input.

10772 Blank lines and those whose first non-<blank> is '#' shall be ignored.

10773 XSI The text files **/usr/lib/cron/cron.allow** and **/usr/lib/cron/cron.deny** shall contain zero or more  
 10774 user names, one per line, of users who are, respectively, authorized or denied access to the  
 10775 service underlying the *crontab* utility.

10776 **ENVIRONMENT VARIABLES**

10777 The following environment variables shall affect the execution of *crontab*:

- |                                  |                 |                                                                                                                                                                                                                                                                                                       |
|----------------------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10778<br>10779                   | <b>EDITOR</b>   | Determine the editor to be invoked when the <b>-e</b> option is specified. The default editor shall be <i>vi</i> .                                                                                                                                                                                    |
| 10780<br>10781<br>10782<br>10783 | <b>LANG</b>     | Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.) |
| 10784<br>10785                   | <b>LC_ALL</b>   | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                              |
| 10786<br>10787<br>10788          | <b>LC_CTYPE</b> | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                                                             |



10789 **LC\_MESSAGES**  
 10790 Determine the locale that should be used to affect the format and contents of  
 10791 diagnostic messages written to standard error.

10792 xsi **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

10793 **ASYNCHRONOUS EVENTS**  
 10794 Default.

10795 **STDOUT**  
 10796 If the **-I** option is specified, the crontab entry shall be written to the standard output.

10797 **STDERR**  
 10798 The standard error shall be used only for diagnostic messages.

10799 **OUTPUT FILES**  
 10800 None.

10801 **EXTENDED DESCRIPTION**  
 10802 None.

10803 **EXIT STATUS**  
 10804 The following exit values shall be returned:  
 10805 0 Successful completion.  
 10806 >0 An error occurred.

10807 **CONSEQUENCES OF ERRORS**  
 10808 The user's crontab entry is not submitted, removed, edited, or listed.

10809 **APPLICATION USAGE**  
 10810 The format of the crontab entry shown here is guaranteed only for the POSIX locale. Other  
 10811 cultures may be supported with substantially different interfaces, although implementations are  
 10812 encouraged to provide comparable levels of functionality.

10813 The default settings of the *HOME*, *LOGNAME*, *PATH*, and *SHELL* variables that are given to the  
 10814 scheduled job are not affected by the settings of those variables when *crontab* is run; as stated,  
 10815 they are defaults. The text about "invoked as specified by this volume of IEEE Std 1003.1-2001"  
 10816 means that the implementation may provide extensions that allow these variables to be affected  
 10817 at runtime, but that the user has to take explicit action in order to access the extension, such as  
 10818 give a new option flag or modify the format of the crontab entry.

10819 A typical user error is to type only *crontab*; this causes the system to wait for the new crontab  
 10820 entry on standard input. If end-of-file is typed (generally <control>-D), the crontab entry is  
 10821 replaced by an empty file. In this case, the user should type the interrupt character, which  
 10822 prevents the crontab entry from being replaced.

10823 **EXAMPLES**

10824 1. Clean up **core** files every weekday morning at 3:15 am:  
 10825 15 3 \* \* 1-5 find \$HOME -name core 2>/dev/null | xargs rm -f

10826 2. Mail a birthday greeting:  
 10827 0 12 14 2 \* mailx john%Happy Birthday!%Time for lunch.

10828 3. As an example of specifying the two types of days:  
 10829 0 0 1,15 \* 1

10830 would run a command on the first and fifteenth of each month, as well as on every  
10831 Monday. To specify days by only one field, the other field should be set to '\*'; for  
10832 example:

10833 0 0 \* \* 1

10834 would run a command only on Mondays.

#### 10835 **RATIONALE**

10836 All references to a *cron* daemon and to *cron files* have been omitted. Although historical  
10837 implementations have used this arrangement, there is no reason to limit future implementations.

10838 This description of *crontab* is designed to support only users with normal privileges. The format  
10839 of the input is based on the System V *crontab*; however, there is no requirement here that the  
10840 actual system database used by the *cron* daemon (or a similar mechanism) use this format  
10841 internally. For example, systems derived from BSD are likely to have an additional field  
10842 appended that indicates the user identity to be used when the job is submitted.

10843 The **-e** option was adopted from the SVID as a user convenience, although it does not exist in all  
10844 historical implementations.

#### 10845 **FUTURE DIRECTIONS**

10846 None.

#### 10847 **SEE ALSO**

10848 *at*

#### 10849 **CHANGE HISTORY**

10850 First released in Issue 2.

#### 10851 **Issue 6**

10852 This utility is marked as part of the User Portability Utilities option.

10853 The normative text is reworded to avoid use of the term “must” for application requirements.

10854 **NAME**

10855           csplit — split files based on context

10856 **SYNOPSIS**10857 UP       csplit [-ks] [-f *prefix*] [-n *number*] *file* *arg1* ...*argn*

10858

10859 **DESCRIPTION**

10860       The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into  
 10861       other files as directed by the *arg* operands, and write the sizes of the files.

10862 **OPTIONS**

10863       The *csplit* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 10864       12.2, Utility Syntax Guidelines.

10865       The following options shall be supported:

10866       **-f** *prefix*     Name the created files *prefix00*, *prefix01*, ..., *prefixn*. The default is **xx00** ... **xxn**. If  
 10867       the *prefix* argument would create a filename exceeding {NAME\_MAX} bytes, an  
 10868       error shall result, *csplit* shall exit with a diagnostic message, and no files shall be  
 10869       created.

10870       **-k**            Leave previously created files intact. By default, *csplit* shall remove created files if  
 10871       an error occurs.

10872       **-n** *number*    Use *number* decimal digits to form filenames for the file pieces. The default shall be  
 10873       2.

10874       **-s**            Suppress the output of file size messages.

10875 **OPERANDS**

10876       The following operands shall be supported:

10877       *file*           The pathname of a text file to be split. If *file* is '-', the standard input shall be  
 10878       used.

10879       The operands *arg1* ... *argn* can be a combination of the following:

10880       /*rexp*/[*offset*]

10881       A file shall be created using the content of the lines from the current line up to, but  
 10882       not including, the line that results from the evaluation of the regular expression  
 10883       with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for  
 10884       basic regular expressions described in the Base Definitions volume of  
 10885       IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions. The application shall  
 10886       use the sequence "\/" to specify a slash character within the *rexp*. The optional  
 10887       offset shall be a positive or negative integer value representing a number of lines.  
 10888       A positive integer value can be preceded by '+'. If the selection of lines from an  
 10889       *offset* expression of this type would create a file with zero lines, or one with greater  
 10890       than the number of lines left in the input file, the results are unspecified. After the  
 10891       section is created, the current line shall be set to the line that results from the  
 10892       evaluation of the regular expression with any offset applied. If the current line is  
 10893       the first line in the file and a regular expression operation has not yet been  
 10894       performed, the pattern match of *rexp* shall be applied from the current line to the  
 10895       end of the file. Otherwise, the pattern match of *rexp* shall be applied from the line  
 10896       following the current line to the end of the file.

10897       %*rexp*%[*offset*]

10898       Equivalent to /*rexp*/[*offset*], except that no file shall be created for the selected  
 10899       section of the input file. The application shall use the sequence "\%" to specify a

10900                   percent-sign character within the *rexp*.

10901           *line\_no*    Create a file from the current line up to (but not including) the line number *line\_no*.  
 10902                   Lines in the file shall be numbered starting at one. The current line becomes  
 10903                   *line\_no*.

10904           {*num*}       Repeat operand. This operand can follow any of the operands described  
 10905                   previously. If it follows a *rexp* type operand, that operand shall be applied *num*  
 10906                   more times. If it follows a *line\_no* operand, the file shall be split every *line\_no* lines,  
 10907                   *num* times, from that point.

10908           An error shall be reported if an operand does not reference a line between the current position  
 10909           and the end of the file.

10910 **STDIN**

10911           See the INPUT FILES section.

10912 **INPUT FILES**

10913           The input file shall be a text file.

10914 **ENVIRONMENT VARIABLES**

10915           The following environment variables shall affect the execution of *csplit*:

10916           *LANG*        Provide a default value for the internationalization variables that are unset or null.  
 10917                   (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 10918                   Internationalization Variables for the precedence of internationalization variables  
 10919                   used to determine the values of locale categories.)

10920           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 10921                   internationalization variables.

10922           *LC\_COLLATE*

10923                   Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 10924                   character collating elements within regular expressions.

10925           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 10926                   characters (for example, single-byte as opposed to multi-byte characters in  
 10927                   arguments and input files) and the behavior of character classes within regular  
 10928                   expressions.

10929           *LC\_MESSAGES*

10930                   Determine the locale that should be used to affect the format and contents of  
 10931                   diagnostic messages written to standard error.

10932 XSI       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

10933 **ASYNCHRONOUS EVENTS**

10934           If the **-k** option is specified, created files shall be retained. Otherwise, the default action occurs.

10935 **STDOUT**

10936           Unless the **-s** option is used, the standard output shall consist of one line per file created, with a  
 10937           format as follows:

10938           "%d\n", <file size in bytes>

10939 **STDERR**

10940           The standard error shall be used only for diagnostic messages.

10941 **OUTPUT FILES**

10942 The output files shall contain portions of the original input file; otherwise, unchanged.

10943 **EXTENDED DESCRIPTION**

10944 None.

10945 **EXIT STATUS**

10946 The following exit values shall be returned:

10947 0 Successful completion.

10948 >0 An error occurred.

10949 **CONSEQUENCES OF ERRORS**

10950 By default, created files shall be removed if an error occurs. When the **-k** option is specified,  
10951 created files shall not be removed if an error occurs.

10952 **APPLICATION USAGE**

10953 None.

10954 **EXAMPLES**

10955 1. This example creates four files, **cobol00** ... **cobol03**:

10956 `csplit -f cobol file '/procedure division/' /par5./ /par16./`

10957 After editing the split files, they can be recombined as follows:

10958 `cat cobol0[0-3] > file`

10959 Note that this example overwrites the original file.

10960 2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up  
10961 to 9999 lines; this is because lines in the file are numbered from 1 rather than zero, for  
10962 historical reasons:

10963 `csplit -k file 100 {99}`

10964 3. Assuming that **prog.c** follows the C-language coding convention of ending routines with a  
10965 `'}'` at the beginning of the line, this example creates a file containing each separate C  
10966 routine (up to 21) in **prog.c**:

10967 `csplit -k prog.c '%main(%' '/^}/+1' {20}`

10968 **RATIONALE**

10969 The **-n** option was added to extend the range of filenames that could be handled.

10970 Consideration was given to adding a **-a** flag to use the alphabetic filename generation used by  
10971 the historical *split* utility, but the functionality added by the **-n** option was deemed to make  
10972 alphabetic naming unnecessary.

10973 **FUTURE DIRECTIONS**

10974 None.

10975 **SEE ALSO**

10976 *sed*, *split*

10977 **CHANGE HISTORY**

10978 First released in Issue 2.

10979 **Issue 5**

10980 The FUTURE DIRECTIONS section is added.

10981 **Issue 6**

10982 This utility is marked as part of the User Portability Utilities option.

10983 The APPLICATION USAGE section is added.

10984 The description of regular expression operands is changed to align with the IEEE P1003.2b draft  
10985 standard.

10986 The normative text is reworded to avoid use of the term “must” for application requirements.

10987 **NAME**10988           ctags — create a tags file (**DEVELOPMENT**, **FORTTRAN**)10989 **SYNOPSIS**10990 UP       ctags [-a] [-f *tagsfile*] *pathname* ...10991           ctags -x *pathname* ...

10992

10993 **DESCRIPTION**

10994       The *ctags* utility shall be provided on systems that support the User Portability Utilities option,  
 10995       the Software Development Utilities option, and either or both of the C-Language Development  
 10996       Utilities option and FORTRAN Development Utilities option. On other systems, it is optional.

10997       The *ctags* utility shall write a *tagsfile* or an index of objects from C-language or FORTRAN source  
 10998       files specified by the *pathname* operands. The *tagsfile* shall list the locators of language-specific  
 10999       objects within the source files. A locator consists of a name, *pathname*, and either a search  
 11000       pattern or a line number that can be used in searching for the object definition. The objects that  
 11001       shall be recognized are specified in the EXTENDED DESCRIPTION section.

11002 **OPTIONS**

11003       The *ctags* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 11004       12.2, Utility Syntax Guidelines.

11005       The following options shall be supported:

11006       **-a**           Append to *tagsfile*.

11007       **-f *tagsfile***   Write the object locator lists into *tagsfile* instead of the default file named **tags** in  
 11008       the current directory.

11009       **-x**           Produce a list of object names, the line number, and filename in which each is  
 11010       defined, as well as the text of that line, and write this to the standard output. A  
 11011       *tagsfile* shall not be created when **-x** is specified.

11012 **OPERANDS**

11013       The following *pathname* operands are supported:

11014       ***file.c***       Files with basenames ending with the **.c** suffix shall be treated as C-language  
 11015       source code. Such files that are not valid input to *c99* produce unspecified results.

11016       ***file.h***       Files with basenames ending with the **.h** suffix shall be treated as C-language  
 11017       source code. Such files that are not valid input to *c99* produce unspecified results.

11018       ***file.f***       Files with basenames ending with the **.f** suffix shall be treated as FORTRAN-  
 11019       language source code. Such files that are not valid input to *fort77* produce  
 11020       unspecified results.

11021       The handling of other files is implementation-defined.

11022 **STDIN**

11023       See the INPUT FILES section.

11024 **INPUT FILES**

11025       The input files shall be text files containing source code in the language indicated by the operand  
 11026       filename suffixes.

11027 **ENVIRONMENT VARIABLES**

11028 The following environment variables shall affect the execution of *ctags*:

11029 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 11030 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 11031 Internationalization Variables for the precedence of internationalization variables  
 11032 used to determine the values of locale categories.)

11033 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 11034 internationalization variables.

11035 *LC\_COLLATE*  
 11036 Determine the order in which output is sorted for the *-x* option. The POSIX locale  
 11037 determines the order in which the *tagsfile* is written.

11038 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 11039 characters (for example, single-byte as opposed to multi-byte characters in  
 11040 arguments and input files). When processing C-language source code, if the locale  
 11041 is not compatible with the C locale described by the ISO C standard, the results are  
 11042 unspecified.

11043 *LC\_MESSAGES*  
 11044 Determine the locale that should be used to affect the format and contents of  
 11045 diagnostic messages written to standard error.

11046 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

11047 **ASYNCHRONOUS EVENTS**

11048 Default.

11049 **STDOUT**

11050 The list of object name information produced by the *-x* option shall be written to standard  
 11051 output in the following format:

11052 "%s %d %s %s", *<object-name>*, *<line-number>*, *<filename>*, *<text>*

11053 where *<text>* is the text of line *<line-number>* of file *<filename>*.

11054 **STDERR**

11055 The standard error shall be used only for diagnostic messages.

11056 **OUTPUT FILES**

11057 When the *-x* option is not specified, the format of the output file shall be:

11058 "%s\t%s\t/%s/\n", *<identifier>*, *<filename>*, *<pattern>*

11059 where *<pattern>* is a search pattern that could be used by an editor to find the defining instance  
 11060 of *<identifier>* in *<filename>* (where *defining instance* is indicated by the declarations listed in the  
 11061 EXTENDED DESCRIPTION).

11062 An optional circumflex ('^') can be added as a prefix to *<pattern>*, and an optional dollar sign  
 11063 can be appended to *<pattern>* to indicate that the pattern is anchored to the beginning (end) of a  
 11064 line of text. Any slash or backslash characters in *<pattern>* shall be preceded by a backslash  
 11065 character. The anchoring circumflex, dollar sign, and escaping backslash characters shall not be  
 11066 considered part of the search pattern. All other characters in the search pattern shall be  
 11067 considered literal characters.



11068 An alternative format is:

11069 "%s\t%s\t%s?\n", <identifier>, <filename>, <pattern>

11070 which is identical to the first format except that slashes in <pattern> shall not be preceded by  
 11071 escaping backslash characters, and question mark characters in <pattern> shall be preceded by  
 11072 backslash characters.

11073 A second alternative format is:

11074 "%s\t%s\t%d\n", <identifier>, <filename>, <lineno>

11075 where <lineno> is a decimal line number that could be used by an editor to find <identifier> in  
 11076 <filename>.

11077 Neither alternative format shall be produced by *ctags* when it is used as described by  
 11078 IEEE Std 1003.1-2001, but the standard utilities that process tags files shall be able to process  
 11079 those formats as well as the first format.

11080 In any of these formats, the file shall be sorted by identifier, based on the collation sequence in  
 11081 the POSIX locale.

11082 **EXTENDED DESCRIPTION**

11083 If the operand identifies C-language source, the *ctags* utility shall attempt to produce an output  
 11084 line for each of the following objects:

- 11085 • Function definitions
- 11086 • Type definitions
- 11087 • Macros with arguments

11088 It may also produce output for any of the following objects:

- 11089 • Function prototypes
- 11090 • Structures
- 11091 • Unions
- 11092 • Global variable definitions
- 11093 • Enumeration types
- 11094 • Macros without arguments
- 11095 • **#define** statements
- 11096 • **#line** statements

11097 Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C  
 11098 programs. The tag formed shall be created by prefixing **M** to the name of the file, with the  
 11099 trailing **.c**, and leading pathname components (if any) removed.

11100 On systems that do not support the C-Language Development Utilities option, *ctags* produces  
 11101 unspecified results for C-language source code files. It should write to standard error a message  
 11102 identifying this condition and cause a non-zero exit status to be produced.

11103 If the operand identifies FORTRAN source, the *ctags* utility shall produce an output line for each  
 11104 function definition. It may also produce output for any of the following objects:

- 11105 • Subroutine definitions
- 11106 • COMMON statements

- 11107           • PARAMETER statements
- 11108           • DATA and BLOCK DATA statements
- 11109           • Statement numbers
- 11110           On systems that do not support the FORTRAN Development Utilities option, *ctags* produces
- 11111           unspecified results for FORTRAN source code files. It should write to standard error a message
- 11112           identifying this condition and cause a non-zero exit status to be produced.
- 11113           It is implementation-defined what other objects (including duplicate identifiers) produce output.
- 11114 **EXIT STATUS**
- 11115           The following exit values shall be returned:
- 11116           0   Successful completion.
- 11117           >0  An error occurred.
- 11118 **CONSEQUENCES OF ERRORS**
- 11119           Default.
- 11120 **APPLICATION USAGE**
- 11121           The output with **-x** is meant to be a simple index that can be written out as an off-line readable
- 11122           function index. If the input files to *ctags* (such as **.c** files) were not created using the same locale
- 11123           as that in effect when *ctags* **-x** is run, results might not be as expected.
- 11124           The description of C-language processing says “attempts to” because the C language can be
- 11125           greatly confused, especially through the use of **#defines**, and this utility would be of no use if
- 11126           the real C preprocessor were run to identify them. The output from *ctags* may be fooled and
- 11127           incorrect for various constructs.
- 11128 **EXAMPLES**
- 11129           None.
- 11130 **RATIONALE**
- 11131           The option list was significantly reduced from that provided by historical implementations. The
- 11132           **-F** option was omitted as redundant, since it is the default. The **-B** option was omitted as being
- 11133           of very limited usefulness. The **-t** option was omitted since the recognition of **typedefs** is now
- 11134           required for C source files. The **-u** option was omitted because the update function was judged
- 11135           to be not only inefficient, but also rarely needed.
- 11136           An early proposal included a **-w** option to suppress warning diagnostics. Since the types of such
- 11137           diagnostics could not be described, the option was omitted as being not useful.
- 11138           The text for *LC\_CTYPE* about compatibility with the C locale acknowledges that the ISO C
- 11139           standard imposes requirements on the locale used to process C source. This could easily be a
- 11140           superset of that known as “the C locale” by way of implementation extensions, or one of a few
- 11141           alternative locales for systems supporting different codesets. No statement is made for
- 11142           FORTRAN because the ANSI X3.9-1978 standard (FORTRAN 77) does not (yet) define a similar
- 11143           locale concept. However, a general rule in this volume of IEEE Std 1003.1-2001 is that any time
- 11144           that locales do not match (preparing a file for one locale and processing it in another), the results
- 11145           are suspect.
- 11146           The collation sequence of the tags file is not affected by *LC\_COLLATE* because it is typically not
- 11147           used by human readers, but only by programs such as *vi* to locate the tag within the source files.
- 11148           Using the POSIX locale eliminates some of the problems of coordinating locales between the
- 11149           *ctags* file creator and the *vi* file reader.

Historically, the tags file has been used only by *ex* and *vi*. However, the format of the tags file has been published to encourage other programs to use the tags in new ways. The format allows either patterns or line numbers to find the identifiers because the historical *vi* recognizes either. The *ctags* utility does not produce the format using line numbers because it is not useful following any source file changes that add or delete lines. The documented search patterns match historical practice. It should be noted that literal leading circumflex or trailing dollar-sign characters in the search pattern will only behave correctly if anchored to the beginning of the line or end of the line by an additional circumflex or dollar-sign character.

Historical implementations also understand the objects used by the languages Pascal and sometimes LISP, and they understand the C source output by *lex* and *yacc*. The *ctags* utility is not required to accommodate these languages, although implementors are encouraged to do so.

The following historical option was not specified, as *vgrind* is not included in this volume of IEEE Std 1003.1-2001:

**-v** If the **-v** flag is given, an index of the form expected by *vgrind* is produced on the standard output. This listing contains the function name, filename, and page number (assuming 64-line pages). Since the output is sorted into lexicographic order, it may be desired to run the output through *sort -f*. Sample use:

```
ctags -v files | sort -f > index vgrind -x index
```

The special treatment of the tag **main** makes the use of *ctags* practical in directories with more than one program.

#### 11170 FUTURE DIRECTIONS

11171 None.

#### 11172 SEE ALSO

11173 *c99*, *fort77*, *vi*

#### 11174 CHANGE HISTORY

11175 First released in Issue 4.

#### 11176 Issue 5

11177 The FUTURE DIRECTIONS section is added.

#### 11178 Issue 6

11179 This utility is marked as part of the User Portability Utilities option.

11180 The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

11181 The normative text is reworded to avoid use of the term “must” for application requirements.

11182 IEEE PASC Interpretation 1003.2 #168 is applied, changing “create” to “write” in the  
11183 DESCRIPTION.

11184 **NAME**

11185           cut — cut out selected fields of each line of a file

11186 **SYNOPSIS**11187           cut -b *list* [-n] [*file* ...]11188           cut -c *list* [*file* ...]11189           cut -f *list* [-d *delim*] [-s] [*file* ...]11190 **DESCRIPTION**

11191           The *cut* utility shall cut out bytes (**-b** option), characters (**-c** option), or character-delimited fields  
 11192           (**-f** option) from each line in one or more files, concatenate them, and write them to standard  
 11193           output.

11194 **OPTIONS**

11195           The *cut* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 11196           12.2, Utility Syntax Guidelines.

11197           The application shall ensure that the option-argument *list* (see options **-b**, **-c**, and **-f** below) is a  
 11198           comma-separated list or <blank>-separated list of positive numbers and ranges. Ranges can be  
 11199           in three forms. The first is two positive numbers separated by a hyphen (*low-high*), which  
 11200           represents all fields from the first number to the second number. The second is a positive  
 11201           number preceded by a hyphen (*-high*), which represents all fields from field number 1 to that  
 11202           number. The third is a positive number followed by a hyphen (*low-*), which represents that  
 11203           number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can be  
 11204           specified in any order, but the bytes, characters, or fields selected shall be written in the order of  
 11205           the input data. If an element appears in the selection list more than once, it shall be written  
 11206           exactly once.

11207           The following options shall be supported:

11208           **-b list**       Cut based on a *list* of bytes. Each selected byte shall be output unless the **-n** option  
 11209                           is also specified. It shall not be an error to select bytes not present in the input line.

11210           **-c list**       Cut based on a *list* of characters. Each selected character shall be output. It shall  
 11211                           not be an error to select characters not present in the input line.

11212           **-d delim**     Set the field delimiter to the character *delim*. The default is the <tab>.

11213           **-f list**       Cut based on a *list* of fields, assumed to be separated in the file by a delimiter  
 11214                           character (see **-d**). Each selected field shall be output. Output fields shall be  
 11215                           separated by a single occurrence of the field delimiter character. Lines with no field  
 11216                           delimiters shall be passed through intact, unless **-s** is specified. It shall not be an  
 11217                           error to select fields not present in the input line.

11218           **-n**           Do not split characters. When specified with the **-b** option, each element in *list* of  
 11219                           the form *low-high* (hyphen-separated numbers) shall be modified as follows:

- 11220                       • If the byte selected by *low* is not the first byte of a character, *low* shall be  
 11221                       decremented to select the first byte of the character originally selected by *low*.  
 11222                       If the byte selected by *high* is not the last byte of a character, *high* shall be  
 11223                       decremented to select the last byte of the character prior to the character  
 11224                       originally selected by *high*, or zero if there is no prior character. If the resulting  
 11225                       range element has *high* equal to zero or *low* greater than *high*, the list element  
 11226                       shall be dropped from *list* for that input line without causing an error.

11227           Each element in *list* of the form *low-* shall be treated as above with *high* set to the  
 11228           number of bytes in the current line, not including the terminating <newline>. Each

11229 element in *list* of the form *–high* shall be treated as above with *low* set to 1. Each  
 11230 element in *list* of the form *num* (a single number) shall be treated as above with *low*  
 11231 set to *num* and *high* set to *num*.

11232 **–s** Suppress lines with no delimiter characters, when used with the **–f** option. Unless  
 11233 specified, lines with no delimiters shall be passed through untouched.

#### 11234 OPERANDS

11235 The following operand shall be supported:

11236 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is  
 11237 '–', the standard input shall be used.

#### 11238 STDIN

11239 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '–'.  
 11240 See the INPUT FILES section.

#### 11241 INPUT FILES

11242 The input files shall be text files, except that line lengths shall be unlimited.

#### 11243 ENVIRONMENT VARIABLES

11244 The following environment variables shall affect the execution of *cut*:

11245 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 11246 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 11247 Internationalization Variables for the precedence of internationalization variables  
 11248 used to determine the values of locale categories.)

11249 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 11250 internationalization variables.

11251 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 11252 characters (for example, single-byte as opposed to multi-byte characters in  
 11253 arguments and input files).

11254 *LC\_MESSAGES*  
 11255 Determine the locale that should be used to affect the format and contents of  
 11256 diagnostic messages written to standard error.

11257 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 11258 ASYNCHRONOUS EVENTS

11259 Default.

#### 11260 STDOUT

11261 The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of  
 11262 the following):

11263 "%s\n", <concatenation of bytes>

11264 "%s\n", <concatenation of characters>

11265 "%s\n", <concatenation of fields and field delimiters>

#### 11266 STDERR

11267 The standard error shall be used only for diagnostic messages.

#### 11268 OUTPUT FILES

11269 None.

11270 **EXTENDED DESCRIPTION**

11271 None.

11272 **EXIT STATUS**

11273 The following exit values shall be returned:

11274 0 All input files were output successfully.

11275 &gt;0 An error occurred.

11276 **CONSEQUENCES OF ERRORS**

11277 Default.

11278 **APPLICATION USAGE**

11279 Earlier versions of the *cut* utility worked in an environment where bytes and characters were  
 11280 considered equivalent (modulo <backspace> and <tab> processing in some implementations). In  
 11281 the extended world of multi-byte characters, the new **-b** option has been added. The **-n** option  
 11282 (used with **-b**) allows it to be used to act on bytes rounded to character boundaries. The  
 11283 algorithm specified for **-n** guarantees that:

11284 `cut -b 1-500 -n file > file1`11285 `cut -b 501- -n file > file2`

11286 ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is,  
 11287 however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

11288 **EXAMPLES**

11289 Examples of the option qualifier list:

11290 1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

11291 1-3,8 Equivalent to 1,2,3,8.

11292 -5,10 Equivalent to 1,2,3,4,5,10.

11293 3- Equivalent to third to last, inclusive.

11294 The *low-high* forms are not always equivalent when used with **-b** and **-n** and multi-byte  
 11295 characters; see the description of **-n**.

11296 The following command:

11297 `cut -d : -f 1,6 /etc/passwd`

11298 reads the System V password file (user database) and produces lines of the form:

11299 `<user ID>:<home directory>`

11300 Most utilities in this volume of IEEE Std 1003.1-2001 work on text files. The *cut* utility can be  
 11301 used to turn files with arbitrary line lengths into a set of text files containing the same data. The  
 11302 *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file**  
 11303 contains long lines:

11304 `cut -b 1-500 -n file > file1`11305 `cut -b 501- -n file > file2`

11306 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that  
 11307 contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in  
 11308 **file** that are longer than 500 + {LINE\_MAX} bytes.) The original file can be recreated from **file1**  
 11309 and **file2** using the command:

11310 `paste -d "\0" file1 file2 > file`

11311 **RATIONALE**

11312 Some historical implementations do not count <backspace>s in determining character counts  
 11313 with the **-c** option. This may be useful for using *cut* for processing *nroff* output. It was  
 11314 deliberately decided not to have the **-c** option treat either <backspace>s or <tab>s in any special  
 11315 fashion. The *fold* utility does treat these characters specially.

11316 Unlike other utilities, some historical implementations of *cut* exit after not finding an input file,  
 11317 rather than continuing to process the remaining *file* operands. This behavior is prohibited by this  
 11318 volume of IEEE Std 1003.1-2001, where only the exit status is affected by this problem.

11319 The behavior of *cut* when provided with either mutually-exclusive options or options that do  
 11320 not work logically together has been deliberately left unspecified in favor of global wording in  
 11321 Section 1.11 (on page 20).

11322 The OPTIONS section was changed in response to IEEE PASC Interpretation 1003.2 #149. The  
 11323 change represents historical practice on all known systems. The original standard was  
 11324 ambiguous on the nature of the output.

11325 The *list* option-arguments are historically used to select the portions of the line to be written, but  
 11326 do not affect the order of the data. For example:

11327 `echo abcdefghi | cut -c6,2,4-7,1`

11328 yields "abdefg".

11329 A proposal to enhance *cut* with the following option:

11330 **-o** Preserve the selected field order. When this option is specified, each byte, character, or field  
 11331 (or ranges of such) shall be written in the order specified by the *list* option-argument, even if  
 11332 this requires multiple outputs of the same bytes, characters, or fields.

11333 was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft  
 11334 standard.

11335 **FUTURE DIRECTIONS**

11336 None.

11337 **SEE ALSO**

11338 *grep*, *paste*, Section 2.5 (on page 33)

11339 **CHANGE HISTORY**

11340 First released in Issue 2.

11341 **Issue 6**

11342 The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

11343 The normative text is reworded to avoid use of the term “must” for application requirements.

11344 **NAME**11345 cxref — generate a C-language program cross-reference table (**DEVELOPMENT**)11346 **SYNOPSIS**

```
11347 xsi cxref [-cs] [-o file] [-w num] [-D name[=def]]... [-I dir]...
11348 [-U name]... file ...
```

11349

11350 **DESCRIPTION**

11351 The *cxref* utility shall analyze a collection of C-language *files* and attempt to build a cross-  
 11352 reference table. Information from **#define** lines shall be included in the symbol table. A sorted  
 11353 listing shall be written to standard output of all symbols (auto, static, and global) in each *file*  
 11354 separately, or with the **-c** option, in combination. Each symbol shall contain an asterisk before  
 11355 the declaring reference.

11356 **OPTIONS**

11357 The *cxref* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 11358 12.2, Utility Syntax Guidelines, except that the order of the **-D**, **-I**, and **-U** options (which are  
 11359 identical to their interpretation by *c99*) is significant. The following options shall be supported:

|       |                |                                                                                         |
|-------|----------------|-----------------------------------------------------------------------------------------|
| 11360 | <b>-c</b>      | Write a combined cross-reference of all input files.                                    |
| 11361 | <b>-s</b>      | Operate silently; do not print input filenames.                                         |
| 11362 | <b>-o file</b> | Direct output to named <i>file</i> .                                                    |
| 11363 | <b>-w num</b>  | Format output no wider than <i>num</i> (decimal) columns. This option defaults to 80 if |
| 11364 |                | <i>num</i> is not specified or is less than 51.                                         |
| 11365 | <b>-D</b>      | Equivalent to <i>c99</i> .                                                              |
| 11366 | <b>-I</b>      | Equivalent to <i>c99</i> .                                                              |
| 11367 | <b>-U</b>      | Equivalent to <i>c99</i> .                                                              |

11368 **OPERANDS**

11369 The following operand shall be supported:

|       |             |                                         |
|-------|-------------|-----------------------------------------|
| 11370 | <i>file</i> | A pathname of a C-language source file. |
|-------|-------------|-----------------------------------------|

11371 **STDIN**

11372 Not used.

11373 **INPUT FILES**

11374 The input files are C-language source files.

11375 **ENVIRONMENT VARIABLES**

11376 The following environment variables shall affect the execution of *cxref*:

|       |                   |                                                                                                                                                                                                                                                                                                                            |
|-------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11377 | <b>LANG</b>       | Provide a default value for the internationalization variables that are unset or null.<br>(See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,<br>11378 Internationalization Variables for the precedence of internationalization variables<br>11379 used to determine the values of locale categories.) |
| 11381 | <b>LC_ALL</b>     | If set to a non-empty string value, override the values of all the other<br>11382 internationalization variables.                                                                                                                                                                                                          |
| 11383 | <b>LC_COLLATE</b> |                                                                                                                                                                                                                                                                                                                            |
| 11384 |                   | Determine the locale for the ordering of the output.                                                                                                                                                                                                                                                                       |
| 11385 | <b>LC_CTYPE</b>   | Determine the locale for the interpretation of sequences of bytes of text data as<br>11386 characters (for example, single-byte as opposed to multi-byte characters in                                                                                                                                                     |



11387 arguments and input files).

11388 **LC\_MESSAGES**

11389 Determine the locale that should be used to affect the format and contents of

11390 diagnostic messages written to standard error.

11391 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

11392 **ASYNCHRONOUS EVENTS**

11393 Default.

11394 **STDOUT**

11395 The standard output shall be used for the cross-reference listing, unless the **-o** option is used to

11396 select a different output file.

11397 The format of standard output is unspecified, except that the following information shall be

11398 included:

- 11399 • If the **-c** option is not specified, each portion of the listing shall start with the name of the
- 11400 input file on a separate line.
- 11401 • The name line shall be followed by a sorted list of symbols, each with its associated location
- 11402 pathname, the name of the function in which it appears (if it is not a function name itself),
- 11403 and line number references.
- 11404 • Each line number may be preceded by an asterisk (**'\*'**) flag, meaning that this is the
- 11405 declaring reference. Other single-character flags, with implementation-defined meanings,
- 11406 may be included.

11407 **STDERR**

11408 The standard error shall be used only for diagnostic messages.

11409 **OUTPUT FILES**

11410 The output file named by the **-o** option shall be used instead of standard output.

11411 **EXTENDED DESCRIPTION**

11412 None.

11413 **EXIT STATUS**

11414 The following exit values shall be returned:

- 11415 0 Successful completion.
- 11416 >0 An error occurred.

11417 **CONSEQUENCES OF ERRORS**

11418 Default.

11419 **APPLICATION USAGE**

11420 None.

11421 **EXAMPLES**

11422 None.

11423 **RATIONALE**

11424 None.

11425 **FUTURE DIRECTIONS**

11426 None.

11427 **SEE ALSO**11428 *c99*11429 **CHANGE HISTORY**

11430 First released in Issue 2.

11431 **Issue 5**11432 In the SYNOPSIS, [**-U** *dir*] is changed to [**-U** *name*].11433 **Issue 6**

11434 The APPLICATION USAGE section is added.

11435 **NAME**

11436           date — write the date and time

11437 **SYNOPSIS**11438           date [-u] [+*format*]11439 xSI       date [-u] *mmddhhmm* [*cc* *yy*]

11440

11441 **DESCRIPTION**

11442 xSI       The *date* utility shall write the date and time to standard output or attempt to set the system date  
 11443           and time. By default, the current date and time shall be written. If an operand beginning with  
 11444           '+' is specified, the output format of *date* shall be controlled by the conversion specifications  
 11445           and other text in the operand.

11446 **OPTIONS**

11447           The *date* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 11448           12.2, Utility Syntax Guidelines.

11449           The following option shall be supported:

11450           **-u**       Perform operations as if the *TZ* environment variable was set to the string "UTC0",  
 11451                       or its equivalent historical value of "GMT0". Otherwise, *date* shall use the  
 11452                       timezone indicated by the *TZ* environment variable or the system default if that  
 11453                       variable is unset or null.

11454 **OPERANDS**

11455           The following operands shall be supported:

11456           **+format**   When the format is specified, each conversion specifier shall be replaced in the  
 11457                       standard output by its corresponding value. All other characters shall be copied to  
 11458                       the output without change. The output shall always be terminated with a  
 11459                       <newline>.

11460           **Conversion Specifications**

|       |    |                                                                          |
|-------|----|--------------------------------------------------------------------------|
| 11461 | %a | Locale's abbreviated weekday name.                                       |
| 11462 | %A | Locale's full weekday name.                                              |
| 11463 | %b | Locale's abbreviated month name.                                         |
| 11464 | %B | Locale's full month name.                                                |
| 11465 | %c | Locale's appropriate date and time representation.                       |
| 11466 | %C | Century (a year divided by 100 and truncated to an integer) as a decimal |
| 11467 |    | number [00,99].                                                          |
| 11468 | %d | Day of the month as a decimal number [01,31].                            |
| 11469 | %D | Date in the format <i>mm/dd/yy</i> .                                     |
| 11470 | %e | Day of the month as a decimal number [1,31] in a two-digit field with    |
| 11471 |    | leading space character fill.                                            |
| 11472 | %h | A synonym for %b.                                                        |
| 11473 | %H | Hour (24-hour clock) as a decimal number [00,23].                        |
| 11474 | %I | Hour (12-hour clock) as a decimal number [01,12].                        |

|       |    |                                                                              |
|-------|----|------------------------------------------------------------------------------|
| 11475 | %j | Day of the year as a decimal number [001,366].                               |
| 11476 | %m | Month as a decimal number [01,12].                                           |
| 11477 | %M | Minute as a decimal number [00,59].                                          |
| 11478 | %n | A <newline>.                                                                 |
| 11479 | %p | Locale's equivalent of either AM or PM.                                      |
| 11480 | %r | 12-hour clock time [01,12] using the AM/PM notation; in the POSIX            |
| 11481 |    | locale, this shall be equivalent to %I:%M:%S %p.                             |
| 11482 | %S | Seconds as a decimal number [00,60].                                         |
| 11483 | %t | A <tab>.                                                                     |
| 11484 | %T | 24-hour clock time [00,23] in the format <i>HH:MM:SS</i> .                   |
| 11485 | %u | Weekday as a decimal number [1,7] (1=Monday).                                |
| 11486 | %U | Week of the year (Sunday as the first day of the week) as a decimal          |
| 11487 |    | number [00,53]. All days in a new year preceding the first Sunday shall be   |
| 11488 |    | considered to be in week 0.                                                  |
| 11489 | %V | Week of the year (Monday as the first day of the week) as a decimal          |
| 11490 |    | number [01,53]. If the week containing January 1 has four or more days in    |
| 11491 |    | the new year, then it shall be considered week 1; otherwise, it shall be the |
| 11492 |    | last week of the previous year, and the next week shall be week 1.           |
| 11493 | %w | Weekday as a decimal number [0,6] (0=Sunday).                                |
| 11494 | %W | Week of the year (Monday as the first day of the week) as a decimal          |
| 11495 |    | number [00,53]. All days in a new year preceding the first Monday shall      |
| 11496 |    | be considered to be in week 0.                                               |
| 11497 | %x | Locale's appropriate date representation.                                    |
| 11498 | %X | Locale's appropriate time representation.                                    |
| 11499 | %y | Year within century [00,99].                                                 |
| 11500 | %Y | Year with century as a decimal number.                                       |
| 11501 | %Z | Timezone name, or no characters if no timezone is determinable.              |
| 11502 | %% | A percent sign character.                                                    |

11503 See the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5, *LC\_TIME*  
 11504 for the conversion specifier values in the POSIX locale.

#### 11505 **Modified Conversion Specifications**

11506 Some conversion specifiers can be modified by the *E* and *O* modifier characters to  
 11507 indicate a different format or specification as specified in the *LC\_TIME* locale  
 11508 description (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5,  
 11509 *LC\_TIME*). If the corresponding keyword (see **era**, **era\_year**, **era\_d\_fmt**, and  
 11510 **alt\_digits** in the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5,  
 11511 *LC\_TIME*) is not specified or not supported for the current locale, the unmodified  
 11512 conversion specifier value shall be used.

11513 %Ec      Locale's alternative appropriate date and time representation.

|       |     |                                                                                                                                           |
|-------|-----|-------------------------------------------------------------------------------------------------------------------------------------------|
| 11514 | %EC | The name of the base year (period) in the locale's alternative representation.                                                            |
| 11515 |     |                                                                                                                                           |
| 11516 | %Ex | Locale's alternative date representation.                                                                                                 |
| 11517 | %EX | Locale's alternative time representation.                                                                                                 |
| 11518 | %Ey | Offset from %EC (year only) in the locale's alternative representation.                                                                   |
| 11519 | %EY | Full alternative year representation.                                                                                                     |
| 11520 | %Od | Day of month using the locale's alternative numeric symbols.                                                                              |
| 11521 | %Oe | Day of month using the locale's alternative numeric symbols.                                                                              |
| 11522 | %OH | Hour (24-hour clock) using the locale's alternative numeric symbols.                                                                      |
| 11523 | %OI | Hour (12-hour clock) using the locale's alternative numeric symbols.                                                                      |
| 11524 | %Om | Month using the locale's alternative numeric symbols.                                                                                     |
| 11525 | %OM | Minutes using the locale's alternative numeric symbols.                                                                                   |
| 11526 | %OS | Seconds using the locale's alternative numeric symbols.                                                                                   |
| 11527 | %Ou | Weekday as a number in the locale's alternative representation (Monday = 1).                                                              |
| 11528 |     |                                                                                                                                           |
| 11529 | %OU | Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.                             |
| 11530 |     |                                                                                                                                           |
| 11531 | %OV | Week number of the year (Monday as the first day of the week, rules corresponding to %V), using the locale's alternative numeric symbols. |
| 11532 |     |                                                                                                                                           |
| 11533 | %Ow | Weekday as a number in the locale's alternative representation (Sunday = 0).                                                              |
| 11534 |     |                                                                                                                                           |
| 11535 | %OW | Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.                             |
| 11536 |     |                                                                                                                                           |
| 11537 | %Oy | Year (offset from %C) in alternative representation.                                                                                      |

|       |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11538 | XSI | <b>mmdhmm[[cc]yy]</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11539 |     | Attempt to set the system date and time from the value given in the operand. This is only possible if the user has appropriate privileges and the system permits the setting of the system date and time. The first <i>mm</i> is the month (number); <i>dd</i> is the day (number); <i>hh</i> is the hour (number, 24-hour system); the second <i>mm</i> is the minute (number); <i>cc</i> is the century and is the first two digits of the year (this is optional); <i>yy</i> is the last two digits of the year and is optional. If century is not specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The current year is the default if <i>yy</i> is omitted. |
| 11540 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11541 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11542 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11543 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11544 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11545 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11546 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11547 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

|       |              |                                                                                                                                                                                                     |
|-------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11548 | <b>Note:</b> | It is expected that in a future version of IEEE Std 1003.1-2001 the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.) |
| 11549 |              |                                                                                                                                                                                                     |
| 11550 |              |                                                                                                                                                                                                     |

|       |              |
|-------|--------------|
| 11551 | <b>STDIN</b> |
| 11552 | Not used.    |

11553 **INPUT FILES**

11554       None.

11555 **ENVIRONMENT VARIABLES**11556       The following environment variables shall affect the execution of *date*:

11557       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 11558                   (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 11559                   Internationalization Variables for the precedence of internationalization variables  
 11560                   used to determine the values of locale categories.)

11561       *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 11562                   internationalization variables.

11563       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 11564                   characters (for example, single-byte as opposed to multi-byte characters in  
 11565                   arguments).

11566       *LC\_MESSAGES*       Determine the locale that should be used to affect the format and contents of  
 11567                   diagnostic messages written to standard error.  
 11568

11569       *LC\_TIME*     Determine the format and contents of date and time strings written by *date*.

11570 xsi       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

11571       *TZ*           Determine the timezone in which the time and date are written, unless the *-u*  
 11572                   option is specified. If the *TZ* variable is unset or null and *-u* is not specified, an  
 11573                   unspecified system default timezone is used.

11574 **ASYNCHRONOUS EVENTS**

11575       Default.

11576 **STDOUT**

11577       When no formatting operand is specified, the output in the POSIX locale shall be equivalent to  
 11578       specifying:

11579       date "+%a %b %e %H:%M:%S %Z %Y"

11580 **STDERR**

11581       The standard error shall be used only for diagnostic messages.

11582 **OUTPUT FILES**

11583       None.

11584 **EXTENDED DESCRIPTION**

11585       None.

11586 **EXIT STATUS**

11587       The following exit values shall be returned:

11588       0   The date was written successfully.

11589       >0   An error occurred.

11590 **CONSEQUENCES OF ERRORS**

11591       Default.

11592 **APPLICATION USAGE**

11593 Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can  
 11594 contain <newline>s in some locales, so it may be difficult to use the format shown in standard  
 11595 output for parsing the output of *date* in those locales.

11596 The range of values for %S extends from 0 to 60 seconds to accommodate the occasional leap  
 11597 second.

11598 Although certain of the conversion specifiers in the POSIX locale (such as the name of the  
 11599 month) are shown with initial capital letters, this need not be the case in other locales. Programs  
 11600 using these fields may need to adjust the capitalization if the output is going to be used at the  
 11601 beginning of a sentence.

11602 The date string formatting capabilities are intended for use in Gregorian-style calendars,  
 11603 possibly with a different starting year (or years). The %x and %c conversion specifications,  
 11604 however, are intended for local representation; these may be based on a different, non-Gregorian  
 11605 calendar.

11606 The %C conversion specification was introduced to allow a fallback for the %EC (alternative year  
 11607 format base year); it can be viewed as the base of the current subdivision in the Gregorian  
 11608 calendar. The century number is calculated as the year divided by 100 and truncated to an  
 11609 integer; it should not be confused with the use of ordinal numbers for centuries (for example,  
 11610 “twenty-first century”). Both the %Ey and %y can then be viewed as the offset from %EC and %C,  
 11611 respectively.

11612 The E and O modifiers modify the traditional conversion specifiers, so that they can always be  
 11613 used, even if the implementation (or the current locale) does not support the modifier.

11614 The E modifier supports alternative date formats, such as the Japanese Emperor’s Era, as long as  
 11615 these are based on the Gregorian calendar system. Extending the E modifiers to other date  
 11616 elements may provide an implementation-defined extension capable of supporting other  
 11617 calendar systems, especially in combination with the O modifier.

11618 The O modifier supports time and date formats using the locale’s alternative numerical symbols,  
 11619 such as Kanji or Hindi digits or ordinal number representation.

11620 Non-European locales, whether they use Latin digits in computational items or not, often have  
 11621 local forms of the digits for use in date formats. This is not totally unknown even in Europe; a  
 11622 variant of dates uses Roman numerals for the months: the third day of September 1991 would be  
 11623 written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking  
 11624 countries, Hindi digits are used. The %d, %e, %H, %I, %m, %S, %U, %w, %W, and %y conversion  
 11625 specifications always return the date and time field in Latin digits (that is, 0 to 9). The %O  
 11626 modifier was introduced to support the use for display purposes of non-Latin digits. In the  
 11627 *LC\_TIME* category in *localedef*, the optional **alt\_digits** keyword is intended for this purpose. As  
 11628 an example, assume the following (partial) *localedef* source:

```
11629 alt_digits "";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \
11630 "IX";"X";"XI";"XII"
11631 d_fmt "%e.%Om.%Y"
```

11632 With the above date, the command:

```
11633 date "+%x"
```

11634 would yield 3.IX.1991. With the same **d\_fmt**, but without the **alt\_digits**, the command would  
 11635 yield 3.9.1991.

11636 **EXAMPLES**

11637 1. The following are input/output examples of *date* used at arbitrary times in the POSIX  
11638 locale:

```
11639 $ date
11640 Tue Jun 26 09:58:10 PDT 1990

11641 $ date "+DATE: %m/%d/%Y%nTIME: %H:%M:%S"
11642 DATE: 11/02/91
11643 TIME: 13:36:16
```

```
11644 $ date "+TIME: %r"
11645 TIME: 01:36:32 PM
```

11646 2. Examples for Denmark, where the default date and time format is %a %d %b %Y %T %Z:

```
11647 $ LANG=da_DK.iso_8859-1 date
11648 ons 02 okt 1991 15:03:32 CET

11649 $ LANG=da_DK.iso_8859-1 \
11650 date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
11651 DATO: onsdag den 2. oktober 1991
11652 KLOKKEN: 15:03:56
```

11653 3. Examples for Germany, where the default date and time format is %a %d.%h.%Y, %T %Z:

```
11654 $ LANG=De_DE.88591 date
11655 Mi 02.Okt.1991, 15:01:21 MEZ

11656 $ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"
11657 DATUM: Mittwoch, 02. Oktober 1991
11658 ZEIT: 15:02:02
```

11659 4. Examples for France, where the default date and time format is %a %d %h %Y %Z %T:

```
11660 $ LANG=Fr_FR.88591 date
11661 Mer 02 oct 1991 MET 15:03:32

11662 $ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"
11663 JOUR: Mercredi 02 octobre 1991
11664 HEURE: 15:03:56
```

11665 **RATIONALE**

11666 Some of the new options for formatting are from the ISO C standard. The **-u** option was  
11667 introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is  
11668 allowed as an equivalent *TZ* value to be compatible with all of the systems using the BSD  
11669 implementation, where this option originated.

11670 The %e format conversion specification (adopted from System V) was added because the ISO C  
11671 standard conversion specifications did not provide any way to produce the historical default  
11672 *date* output during the first nine days of any month.

11673 There are two varieties of day and week numbering supported (in addition to any others created  
11674 with the locale-dependent %E and %O modifier characters):

- 11675 • The historical variety in which Sunday is the first day of the week and the weekdays
- 11676 preceding the first Sunday of the year are considered week 0. These are represented by %w
- 11677 and %U. A variant of this is %W, using Monday as the first day of the week, but still referring
- 11678 to week 0. This view of the calendar was retained because so many historical applications
- 11679 depend on it and the ISO C standard *strftime()* function, on which many *date*



11680 implementations are based, was defined in this way.

11681 • The international standard, based on the ISO 8601:2000 standard where Monday is the first  
 11682 weekday and the algorithm for the first week number is more complex: If the week (Monday  
 11683 to Sunday) containing January 1 has four or more days in the new year, then it is week 1;  
 11684 otherwise, it is week 53 of the previous year, and the next week is week 1. These are  
 11685 represented by the new conversion specifications %u and %V, added as a result of  
 11686 international comments.

11687 **FUTURE DIRECTIONS**

11688 None.

11689 **SEE ALSO**

11690 The System Interfaces volume of IEEE Std 1003.1-2001, *printf()*, *strptime()*

11691 **CHANGE HISTORY**

11692 First released in Issue 2.

11693 **Issue 5**

11694 Changes are made for Year 2000 alignment.

11695 **Issue 6**

11696 The following new requirements on POSIX implementations derive from alignment with the  
 11697 Single UNIX Specification:

11698 • The %EX modified conversion specification is added. |

11699 The Open Group Corrigendum U048/2 is applied, correcting the examples.

11700 The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors  
 11701 for consistency with the *LC\_TIME* category.

11702 A clarification is made such that the current year is the default if the yy argument is omitted  
 11703 when setting the system date and time.

11704 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/19 is applied, correcting the CHANGE |  
 11705 HISTORY section. |

11706 **NAME**

11707 dd — convert and copy a file

11708 **SYNOPSIS**11709 dd [*operand* ...]11710 **DESCRIPTION**

11711 The *dd* utility shall copy the specified input file to the specified output file with possible  
 11712 conversions using specific input and output block sizes. It shall read the input one block at a  
 11713 time, using the specified input block size; it shall then process the block of data actually  
 11714 returned, which could be smaller than the requested block size. It shall apply any conversions  
 11715 that have been specified and write the resulting data to the output in blocks of the specified  
 11716 output block size. If the **bs=expr** operand is specified and no conversions other than **sync**,  
 11717 **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a  
 11718 separate output block; if the read returns less than a full block and the **sync** conversion is not  
 11719 specified, the resulting output block shall be the same size as the input block. If the **bs=expr**  
 11720 operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the  
 11721 input shall be processed and collected into full-sized output blocks until the end of the input is  
 11722 reached.

11723 The processing order shall be as follows:

- 11724 1. An input block is read.
- 11725 2. If the input block is shorter than the specified input block size and the **sync** conversion is  
 11726 specified, null bytes shall be appended to the input data up to the specified size. (If either  
 11727 **block** or **unblock** is also specified, <space>s shall be appended instead of null bytes.) The  
 11728 remaining conversions and output shall include the pad characters as if they had been read  
 11729 from the input.
- 11730 3. If the **bs=expr** operand is specified and no conversion other than **sync** or **noerror** is  
 11731 requested, the resulting data shall be written to the output as a single block, and the  
 11732 remaining steps are omitted.
- 11733 4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If there  
 11734 is an odd number of bytes in the input block, the last byte in the input record shall not be  
 11735 swapped.
- 11736 5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These  
 11737 conversions shall operate on the input data independently of the input blocking; an input  
 11738 or output fixed-length record may span block boundaries.
- 11739 6. The data resulting from input or conversion or both shall be aggregated into output blocks  
 11740 of the specified size. After the end of input is reached, any remaining output shall be  
 11741 written as a block without padding if **conv=sync** is not specified; thus, the final output  
 11742 block may be shorter than the output block size.

11743 **OPTIONS**

11744 None.

11745 **OPERANDS**

11746 All of the operands shall be processed before any input is read. The following operands shall be  
 11747 supported:

- |       |                |                                                                                                     |
|-------|----------------|-----------------------------------------------------------------------------------------------------|
| 11748 | <b>if=file</b> | Specify the input pathname; the default is standard input.                                          |
| 11749 | <b>of=file</b> | Specify the output pathname; the default is standard output. If the <b>seek=expr</b>                |
| 11750 |                | conversion is not also specified, the output file shall be truncated before the copy                |
| 11751 |                | begins if an explicit <b>of=file</b> operand is specified, unless <b>conv=notrunc</b> is specified. |

|           |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11752     |                               | If <b>seek=expr</b> is specified, but <b>conv=notrunc</b> is not, the effect of the copy shall be to preserve the blocks in the output file over which <i>dd</i> seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy.)                                                                                                                                                                                                                                               |
| 11753     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11754     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11755     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11756     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11757     | <b>ibs=expr</b>               | Specify the input block size, in bytes, by <i>expr</i> (default is 512).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 11758     | <b>obs=expr</b>               | Specify the output block size, in bytes, by <i>expr</i> (default is 512).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11759     | <b>bs=expr</b>                | Set both input and output block sizes to <i>expr</i> bytes, superseding <b>ibs=</b> and <b>obs=</b> . If no conversion other than <b>sync</b> , <b>noerror</b> , and <b>notrunc</b> is specified, each input block shall be copied to the output as a single block without aggregating short blocks.                                                                                                                                                                                                                                                                                                                                           |
| 11760     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11761     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11762     | <b>cbs=expr</b>               | Specify the conversion block size for <b>block</b> and <b>unblock</b> in bytes by <i>expr</i> (default is zero). If <b>cbs=</b> is omitted or given a value of zero, using <b>block</b> or <b>unblock</b> produces unspecified results.                                                                                                                                                                                                                                                                                                                                                                                                        |
| 11763     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11764     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11765 XSI |                               | The application shall ensure that this operand is also specified if the <b>conv=</b> operand is specified with a value of <b>ascii</b> , <b>ebcdic</b> , or <b>ibm</b> . For a <b>conv=</b> operand with an <b>ascii</b> value, the input is handled as described for the <b>unblock</b> value, except that characters are converted to ASCII before any trailing <space>s are deleted. For <b>conv=</b> operands with <b>ebcdic</b> or <b>ibm</b> values, the input is handled as described for the <b>block</b> value except that the characters are converted to EBCDIC or IBM EBCDIC, respectively, after any trailing <space>s are added. |
| 11766     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11767     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11768     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11769     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11770     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11771     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11772     | <b>skip=n</b>                 | Skip <i>n</i> input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on non-seekable files, the blocks shall be read and the data shall be discarded.                                                                                                                                                                                                                                                                                                                                                                                     |
| 11773     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11774     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11775     | <b>seek=n</b>                 | Skip <i>n</i> blocks (using the specified output block size) from the beginning of the output file before copying. On non-seekable files, existing blocks shall be read and space from the current end-of-file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for non-seekable files.                                                                                                                                                                                                                                       |
| 11776     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11777     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11778     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11779     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11780     | <b>count=n</b>                | Copy only <i>n</i> input blocks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 11781     | <b>conv=value[,value ...]</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11782     |                               | Where <i>values</i> are comma-separated symbols from the following list:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 11783 XSI | <b>ascii</b>                  | Convert EBCDIC to ASCII; see Table 4-6 (on page 305).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 11784 XSI | <b>ebcdic</b>                 | Convert ASCII to EBCDIC; see Table 4-6 (on page 305).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 11785 XSI | <b>ibm</b>                    | Convert ASCII to a different EBCDIC set; see Table 4-7 (on page 306).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 11786     |                               | The <b>ascii</b> , <b>ebcdic</b> , and <b>ibm</b> values are mutually-exclusive.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 11787     | <b>block</b>                  | Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space>s shall be appended to lines that are shorter than their conversion block size to fill the block. Lines that are longer than the conversion block size shall be truncated to the largest number of characters that fit into that size; the number of truncated lines shall be reported (see the STDERR section).          |
| 11788     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11789     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11790     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11791     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11792     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11793     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11794     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11795     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11796     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

|           |                                                                                                                |                                                                                                                       |
|-----------|----------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| 11797     |                                                                                                                | The <b>block</b> and <b>unblock</b> values are mutually-exclusive.                                                    |
| 11798     | <b>unblock</b>                                                                                                 | Convert fixed-length records to variable length. Read a number of                                                     |
| 11799     |                                                                                                                | bytes equal to the conversion block size (or the number of bytes                                                      |
| 11800     |                                                                                                                | remaining in the input, if less than the conversion block size), delete                                               |
| 11801     |                                                                                                                | all trailing <space>s, and append a <newline>.                                                                        |
| 11802     | <b>lcase</b>                                                                                                   | Map uppercase characters specified by the <i>LC_CTYPE</i> keyword                                                     |
| 11803     |                                                                                                                | <b>tolower</b> to the corresponding lowercase character. Characters for                                               |
| 11804     |                                                                                                                | which no mapping is specified shall not be modified by this                                                           |
| 11805     |                                                                                                                | conversion.                                                                                                           |
| 11806     |                                                                                                                | The <b>lcase</b> and <b>ucase</b> symbols are mutually-exclusive.                                                     |
| 11807     | <b>ucase</b>                                                                                                   | Map lowercase characters specified by the <i>LC_CTYPE</i> keyword                                                     |
| 11808     |                                                                                                                | <b>toupper</b> to the corresponding uppercase character. Characters for                                               |
| 11809     |                                                                                                                | which no mapping is specified shall not be modified by this                                                           |
| 11810     |                                                                                                                | conversion.                                                                                                           |
| 11811     | <b>swab</b>                                                                                                    | Swap every pair of input bytes.                                                                                       |
| 11812     | <b>noerror</b>                                                                                                 | Do not stop processing on an input error. When an input error                                                         |
| 11813     |                                                                                                                | occurs, a diagnostic message shall be written on standard error,                                                      |
| 11814     |                                                                                                                | followed by the current input and output block counts in the same                                                     |
| 11815     |                                                                                                                | format as used at completion (see the <i>STDERR</i> section). If the <b>sync</b>                                      |
| 11816     |                                                                                                                | conversion is specified, the missing input shall be replaced with null                                                |
| 11817     |                                                                                                                | bytes and processed normally; otherwise, the input block shall be                                                     |
| 11818     |                                                                                                                | omitted from the output.                                                                                              |
| 11819     | <b>notrunc</b>                                                                                                 | Do not truncate the output file. Preserve blocks in the output file not                                               |
| 11820     |                                                                                                                | explicitly written by this invocation of the <i>dd</i> utility. (See also the                                         |
| 11821     |                                                                                                                | preceding <b>of=file</b> operand.)                                                                                    |
| 11822     | <b>sync</b>                                                                                                    | Pad every input block to the size of the <b>ibs=</b> buffer, appending null                                           |
| 11823     |                                                                                                                | bytes. (If either <b>block</b> or <b>unblock</b> is also specified, append <space>s,                                  |
| 11824     |                                                                                                                | rather than null bytes.)                                                                                              |
| 11825     |                                                                                                                | The behavior is unspecified if operands other than <b>conv=</b> are specified more than once.                         |
| 11826     |                                                                                                                | For the <b>bs=</b> , <b>cbs=</b> , <b>ibs=</b> , and <b>obs=</b> operands, the application shall supply an expression |
| 11827     |                                                                                                                | specifying a size in bytes. The expression, <i>expr</i> , can be:                                                     |
| 11828     | 1.                                                                                                             | A positive decimal number                                                                                             |
| 11829     | 2.                                                                                                             | A positive decimal number followed by <i>k</i> , specifying multiplication by 1 024                                   |
| 11830     | 3.                                                                                                             | A positive decimal number followed by <i>b</i> , specifying multiplication by 512                                     |
| 11831     | 4.                                                                                                             | Two or more positive decimal numbers (with or without <i>k</i> or <i>b</i> ) separated by <i>x</i> , specifying       |
| 11832     |                                                                                                                | the product of the indicated values                                                                                   |
| 11833     |                                                                                                                | All of the operands are processed before any input is read.                                                           |
| 11834 xsi | The following two tables display the octal number character values used for the <b>ascii</b> and <b>ebcdic</b> |                                                                                                                       |
| 11835     | conversions (first table) and for the <b>ibm</b> conversion (second table). In both tables, the ASCII          |                                                                                                                       |
| 11836     | values are the row and column headers and the EBCDIC values are found at their intersections.                  |                                                                                                                       |
| 11837     | For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The                     |                                                                                                                       |
| 11838     | inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one                          |                                                                                                                       |
| 11839     | correspondence with these tables. The differences between the two tables are highlighted by                    |                                                                                                                       |
| 11840     | small boxes drawn around five entries.                                                                         |                                                                                                                       |

11841  
11842

Table 4-6 ASCII to EBCDIC Conversion

|      | 0   | 1   | 2   | 3   | 4   | 5   | 6    | 7    |
|------|-----|-----|-----|-----|-----|-----|------|------|
| 0000 | NUL | SOH | STX | ETX | EOT | ENQ | ACK  | BEL  |
| 0010 | BS  | HT  | LF  | VT  | FF  | CR  | SO   | SI   |
| 0020 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN  | ETB  |
| 0030 | CAN | EM  | SUB | ESC | IFS | IGS | IRS  | ITB  |
| 0040 | Sp  | !   | "   | #   | \$  | %   | &    | .    |
| 0050 | (   | )   | *   | +   | ,   | -   | .    | /    |
| 0060 | 0   | 1   | 2   | 3   | 4   | 5   | 6    | 7    |
| 0070 | 8   | 9   | :   | ;   | <   | =   | >    | ?    |
| 0100 | @   | A   | B   | C   | D   | E   | F    | G    |
| 0110 | H   | I   | J   | K   | L   | M   | N    | O    |
| 0120 | P   | Q   | R   | S   | T   | U   | V    | W    |
| 0130 | X   | Y   | Z   | [   | \   | ]   | ^    | _    |
| 0140 | ,   | a   | b   | c   | d   | e   |      |      |
| 0150 | h   | i   | j   | k   | l   | m   | n    | o    |
| 0160 | p   | q   | r   | s   | t   | u   | v    | w    |
| 0170 | x   | y   | z   | {   |     | }   | ~    | DEL  |
| 0200 | DS  | SOS | FS  | WUS | BYP | NL  |      |      |
| 0210 | SA  | SFE | SM  | CSP | MFA | SPS | RNL  | POC  |
| 0220 |     |     | UBS | IR  | PP  | TRN | RPT  | CU1  |
| 0230 | SBS | IT  | RFF | CU3 | SEL | RES | NBS  | GE   |
| 0240 |     |     |     |     |     |     | 0076 | 0341 |
| 0250 |     |     |     |     |     |     | 0107 | 0110 |
| 0260 |     |     |     |     |     |     | 0126 | 0127 |
| 0270 |     |     |     |     |     |     | 0146 | 0147 |
|      |     |     |     |     |     |     | 0164 | 0165 |
| 0300 |     |     |     |     |     |     | 0214 | 0215 |
| 0310 |     |     |     |     |     |     | 0235 | 0236 |
| 0320 |     |     |     |     |     |     | 0256 | 0257 |
| 0330 |     |     |     |     |     |     | 0266 | 0267 |
| 0340 |     |     |     |     |     |     | 0276 | 0277 |
| 0350 |     |     |     |     |     |     | 0332 | 0333 |
| 0360 |     |     |     |     |     |     | 0354 | 0355 |
| 0370 |     |     |     |     |     |     | 0376 | EO   |

11843

**Table 4-7** ASCII to IBM EBCDIC Conversion

11844

|      | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| 0000 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL |
| 0010 | BS  | HT  | LF  | VT  | FF  | CR  | SO  | SI  |
| 0020 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB |
| 0030 | CAN | EM  | SUB | ESC | IFS | IGS | IRS | ITB |
| 0040 | Sp  | !   | "   | #   | \$  | %   | &   | .   |
| 0050 | (   | )   | *   | +   | ,   | -   | .   | /   |
| 0060 | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
| 0070 | 8   | 9   | :   | ;   | <   | =   | >   | ?   |
| 0100 | @   | A   | B   | C   | D   | E   | F   | G   |
| 0110 | H   | I   | J   | K   | L   | M   | N   | O   |
| 0120 | P   | Q   | R   | S   | T   | U   | V   | W   |
| 0130 | X   | Y   | Z   | [   | \   | ]   | ^   | _   |
| 0140 | ,   | a   | b   | c   | d   | e   | f   | g   |
| 0150 | h   | i   | j   | k   | l   | m   | n   | o   |
| 0160 | p   | q   | r   | s   | t   | u   | v   | w   |
| 0170 | x   | y   | z   | {   |     | }   | ~   | DEL |
| 0200 | DS  | SOS | FS  | WUS | BYP | NL  | RNL | POC |
| 0210 | SA  | SFE | SM  | CSP | MFA | SPS | RPT | CU1 |
| 0220 |     |     | UBS | IR  | PP  | TRN | NBS | GE  |
| 0230 | SBS | IT  | RFF | CU3 | SEL | RES |     |     |
| 0240 |     |     |     |     |     |     |     |     |
| 0250 |     |     |     |     |     |     |     |     |
| 0260 |     |     |     |     |     |     |     |     |
| 0270 |     |     |     |     |     |     |     |     |
| 0300 |     |     |     |     |     |     |     |     |
| 0310 |     |     |     |     |     |     |     |     |
| 0320 |     |     |     |     |     |     |     |     |
| 0330 |     |     |     |     |     |     |     |     |
| 0340 |     |     |     |     |     |     |     |     |
| 0350 |     |     |     |     |     |     |     |     |
| 0360 |     |     |     |     |     |     |     |     |
| 0370 |     |     |     |     |     |     |     |     |

11845 **STDIN**

11846 If no **if=** operand is specified, the standard input shall be used. See the INPUT FILES section.

11847 **INPUT FILES**

11848 The input file can be any file type.

11849 **ENVIRONMENT VARIABLES**

11850 The following environment variables shall affect the execution of *dd*:

11851 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 11852 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 11853 Internationalization Variables for the precedence of internationalization variables  
 11854 used to determine the values of locale categories.)

11855 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 11856 internationalization variables.

11857 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 11858 characters (for example, single-byte as opposed to multi-byte characters in  
 11859 arguments and input files), the classification of characters as uppercase or  
 11860 lowercase, and the mapping of characters from one case to the other.

11861 *LC\_MESSAGES*

11862 Determine the locale that should be used to affect the format and contents of  
 11863 diagnostic messages written to standard error and informative messages written to  
 11864 standard output.

11865 *XS1* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

11866 **ASYNCHRONOUS EVENTS**

11867 For SIGINT, the *dd* utility shall interrupt its current processing, write status information to  
 11868 standard error, and exit as though terminated by SIGINT. It shall take the standard action for all  
 11869 other signals; see the ASYNCHRONOUS EVENTS section in Section 1.11 (on page 20).

11870 **STDOUT**

11871 If no **of=** operand is specified, the standard output shall be used. The nature of the output  
 11872 depends on the operands selected.

11873 **STDERR**

11874 On completion, *dd* shall write the number of input and output blocks to standard error. In the  
 11875 POSIX locale the following formats shall be used:

11876 "%u+%u records in\n", <number of whole input blocks>,  
 11877 <number of partial input blocks>

11878 "%u+%u records out\n", <number of whole output blocks>,  
 11879 <number of partial output blocks>

11880 A partial input block is one for which *read()* returned less than the input block size. A partial  
 11881 output block is one that was written with fewer bytes than specified by the output block size.

11882 In addition, when there is at least one truncated block, the number of truncated blocks shall be  
 11883 written to standard error. In the POSIX locale, the format shall be:

11884 "%u truncated %s\n", <number of truncated blocks>, "record" (if  
 11885 <number of truncated blocks> is one) "records" (otherwise)

11886 Diagnostic messages may also be written to standard error.

11887 **OUTPUT FILES**

11888 If the **of=** operand is used, the output shall be the same as described in the STDOUT section.

11889 **EXTENDED DESCRIPTION**

11890 None.

11891 **EXIT STATUS**

11892 The following exit values shall be returned:

11893 0 The input file was copied successfully.

11894 >0 An error occurred.

11895 **CONSEQUENCES OF ERRORS**

11896 If an input error is detected and the **noerror** conversion has not been specified, any partial  
 11897 output block shall be written to the output file, a diagnostic message shall be written, and the  
 11898 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall  
 11899 be written and the copy operation shall be discontinued.

11900 **APPLICATION USAGE**

11901 The input and output block size can be specified to take advantage of raw physical I/O.

11902 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions  
 11903 specified for the *dd* utility perform conversions for the version specified by the tables.

11904 **EXAMPLES**

11905 The following command:

11906 `dd if=/dev/rmt0h of=/dev/rmt1h`

11907 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

11908 The following command:

11909 `dd ibs=10 skip=1`

11910 strips the first 10 bytes from standard input.

11911 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the  
 11912 ASCII file **x**:

11913 `dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase`

11914 **RATIONALE**

11915 The OPTIONS section is listed as “None” because there are no options recognized by historical  
 11916 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax  
 11917 Guidelines, which would have resulted in the classic hyphenated option letters. In this version  
 11918 of this volume of IEEE Std 1003.1-2001, *dd* retains its curious JCL-like syntax due to the large  
 11919 number of applications that depend on the historical implementation.

11920 A suggested implementation technique for **conv=noerror, sync** is to zero (or <space>-fill, if  
 11921 **blocking** or **unblocking**) the input buffer before each read and to write the contents of the input  
 11922 buffer to the output even after an error. In this manner, any data transferred to the input buffer  
 11923 before the error was detected is preserved. Another point is that a failed read on a regular file or  
 11924 a disk generally does not increment the file offset, and *dd* must then seek past the block on which  
 11925 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic  
 11926 tape, however, the tape normally has passed the block containing the error when the error is  
 11927 reported, and thus no seek is necessary.

11928 The default **ibs=** and **obs=** sizes are specified as 512 bytes because there are historical (largely  
 11929 portable) scripts that assume these values. If they were left unspecified, unusual results could



- 11930 occur if an implementation chose an odd block size.
- 11931 Historical implementations of *dd* used *creat()* when processing **of=file**. This makes the **seek=**  
 11932 operand unusable except on special files. The **conv=notrunc** feature was added because more  
 11933 recent BSD-based implementations use *open()* (without *O\_TRUNC*) instead of *creat()*, but they  
 11934 fail to delete output file contents after the data copied.
- 11935 The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to  
 11936 mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of  
 11937 IEEE Std 1003.1-2001.
- 11938 Standard EBCDIC does not have the characters ' [ ' and ' ] '. The values used in the table are  
 11939 taken from a common print train that does contain them. Other than those characters, the print  
 11940 train values are not filled in, but appear to provide some of the motivation for the historical  
 11941 choice of translations reflected here.
- 11942 The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.
- 11943 The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ  
 11944 in such a way that:
- 11945 1. EBCDIC 0112 ( ' ¢ ' ) and 0152 (broken pipe) do not appear in the table.
  - 11946 2. EBCDIC 0137 ( ' ¬ ' ) translates to/from ASCII 0236 ( ' ^ ' ). In the standard table, EBCDIC  
 11947 0232 (no graphic) is used.
  - 11948 3. EBCDIC 0241 ( ' ~ ' ) translates to/from ASCII 0176 ( ' ~ ' ). In the standard table, EBCDIC  
 11949 0137 ( ' ¬ ' ) is used.
  - 11950 4. 0255 ( ' [ ' ) and 0275 ( ' ] ' ) appear twice, once in the same place as for the standard table  
 11951 and once in place of 0112 ( ' ¢ ' ) and 0241 ( ' ~ ' ).
- 11952 In net result:
- 11953 EBCDIC 0275 ( ' ] ' ) displaced EBCDIC 0241 ( ' ~ ' ) in cell 0345.
- 11954 That displaced EBCDIC 0137 ( ' ¬ ' ) in cell 0176.
- 11955 That displaced EBCDIC 0232 (no graphic) in cell 0136.
- 11956 That replaced EBCDIC 0152 (broken pipe) in cell 0313.
- 11957 EBCDIC 0255 ( ' [ ' ) replaced EBCDIC 0112 ( ' ¢ ' ).
- 11958 This translation, however, reflects historical practice that (ASCII) ' ~ ' and ' ¬ ' were often  
 11959 mapped to each other, as were ' [ ' and ' ¢ ' ; and ' ] ' and (EBCDIC) ' ~ ' .
- 11960 The **cbs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the  
 11961 **ascii** operand, the input is handled as described for the **unblock** operand except that characters  
 11962 are converted to ASCII before the trailing <space>s are deleted. For the **ebcdic** and **ibm**  
 11963 operands, the input is handled as described for the **block** operand except that the characters are  
 11964 converted to EBCDIC or IBM EBCDIC after the trailing <space>s are added.
- 11965 The **block** and **unblock** keywords are from historical BSD practice.
- 11966 The consistent use of the word **record** in standard error messages matches most historical  
 11967 practice. An earlier version of System V used **block**, but this has been updated in more recent  
 11968 releases.
- 11969 Early proposals only allowed two numbers separated by **x** to be used in a product when  
 11970 specifying **bs=**, **cbs=**, **ibs=**, and **obs=** sizes. This was changed to reflect the historical practice of  
 11971 allowing multiple numbers in the product as provided by Version 7 and all releases of System V

- 11972 and BSD.
- 11973 A change to the **swab** conversion is required to match historical practice and is the result of IEEE  
11974 PASC Interpretations 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.
- 11975 A change to the handling of SIGINT is required to match historical practice and is the result of  
11976 IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.
- 11977 **FUTURE DIRECTIONS**
- 11978 None.
- 11979 **SEE ALSO**
- 11980 Section 1.11 (on page 20), *sed*, *tr*
- 11981 **CHANGE HISTORY**
- 11982 First released in Issue 2.
- 11983 **Issue 5**
- 11984 The second paragraph of the **cbs=** description is reworded and marked EX.
- 11985 The FUTURE DIRECTIONS section is added.
- 11986 **Issue 6**
- 11987 Changes are made to **swab** conversion and SIGINT handling to align with the IEEE P1003.2b  
11988 draft standard.
- 11989 The normative text is reworded to avoid use of the term “must” for application requirements.
- 11990 IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between *dd of=file* and  
11991 **conv=notrunc**.

11992 **NAME**11993           delta — make a delta (change) to an SCCS file (**DEVELOPMENT**)11994 **SYNOPSIS**11995 xsi       delta [-nps] [-g *list*] [-m *mrlist*] [-r *SID*] [-y[*comment*]] *file...*

11996

11997 **DESCRIPTION**

11998       The *delta* utility shall be used to permanently introduce into the named SCCS files changes that  
 11999       were made to the files retrieved by *get* (called the *g-files*, or generated files).

12000 **OPTIONS**

12001       The *delta* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 12002       12.2, Utility Syntax Guidelines, except that the **-y** option has an optional option-argument. This  
 12003       optional option-argument shall not be presented as a separate argument.

12004       The following options shall be supported:

12005       **-r *SID***       Uniquely identify which delta is to be made to the SCCS file. The use of this option  
 12006       shall be necessary only if two or more outstanding *get* commands for editing (*get*  
 12007       **-e**) on the same SCCS file were done by the same person (login name). The *SID*  
 12008       value specified with the **-r** option can be either the *SID* specified on the *get*  
 12009       command line or the *SID* to be made as reported by the *get* utility; see *get* (on page  
 12010       476).

12011       **-s**            Suppress the report to standard output of the activity associated with each *file*.  
 12012       See the **STDOUT** section.

12013       **-n**            Specify retention of the edited *g-file* (normally removed at completion of delta  
 12014       processing).

12015       **-g *list***       Specify a *list* (see *get* for the definition of *list*) of deltas that shall be ignored when  
 12016       the file is accessed at the change level (*SID*) created by this delta.

12017       **-m *mrlist***   Specify a modification request (MR) number that the application shall supply as  
 12018       the reason for creating the new delta. This shall be used if the SCCS file has the **v**  
 12019       flag set; see *admin*.

12020       If **-m** is not used and **'-'** is not specified as a file argument, and the standard  
 12021       input is a terminal, the prompt described in the **STDOUT** section shall be written  
 12022       to standard output before the standard input is read; if the standard input is not a  
 12023       terminal, no prompt shall be issued.

12024       MRs in a list shall be separated by <blank>s or escaped <newline>s. An  
 12025       unescape <newline> shall terminate the MR list. The escape character is  
 12026       <backslash>.

12027       If the **v** flag has a value, it shall be taken to be the name of a program which  
 12028       validates the correctness of the MR numbers. If a non-zero exit status is returned  
 12029       from the MR number validation program, the *delta* utility shall terminate. (It is  
 12030       assumed that the MR numbers were not all valid.)

12031       **-y[*comment*]** Describe the reason for making the delta. The *comment* shall be an arbitrary group  
 12032       of lines that would meet the definition of a text file. Implementations shall support  
 12033       *comments* from zero to 512 bytes and may support longer values. A null string  
 12034       (specified as either **-y**, **-y" "**, or in response to a prompt for a comment) shall be  
 12035       considered a valid *comment*.

12036 If **-y** is not specified and **'-'** is not specified as a file argument, and the standard  
 12037 input is a terminal, the prompt described in the STDOUT section shall be written  
 12038 to standard output before the standard input is read; if the standard input is not a  
 12039 terminal, no prompt shall be issued. An unescaped **<newline>** shall terminate the  
 12040 comment text. The escape character is **<backslash>**.

12041 The **-y** option shall be required if the *file* operand is specified as **'-'**.

12042 **-p** Write (to standard output) the SCCS file differences before and after the delta is  
 12043 applied in *diff* format; see *diff*.

#### 12044 OPERANDS

12045 The following operand shall be supported:

12046 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta*  
 12047 utility shall behave as though each file in the directory were specified as a named  
 12048 file, except that non-SCCS files (last component of the pathname does not begin  
 12049 with **s.**) and unreadable files shall be silently ignored.

12050 If exactly one *file* operand appears, and it is **'-'**, the standard input shall be read;  
 12051 each line of the standard input shall be taken to be the name of an SCCS file to be  
 12052 processed. Non-SCCS files and unreadable files shall be silently ignored.

#### 12053 STDIN

12054 The standard input shall be a text file used only in the following cases:

- 12055 • To read an *mrlist* or a *comment* (see the **-m** and **-y** options).
- 12056 • A *file* operand shall be specified as **'-'**. In this case, the **-y** option must be used to specify  
 12057 the comment, and if the SCCS file has the **v** flag set, the **-m** option must also be used to  
 12058 specify the MR list.

#### 12059 INPUT FILES

12060 Input files shall be text files whose data is to be included in the SCCS files. If the first character of  
 12061 any line of an input file is **<SOH>** in the POSIX locale, the results are unspecified. If this file  
 12062 contains more than 99 999 lines, the number of lines recorded in the header for this file shall be  
 12063 99 999 for this delta.

#### 12064 ENVIRONMENT VARIABLES

12065 The following environment variables shall affect the execution of *delta*:

12066 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 12067 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 12068 Internationalization Variables for the precedence of internationalization variables  
 12069 used to determine the values of locale categories.)

12070 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 12071 internationalization variables.

12072 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 12073 characters (for example, single-byte as opposed to multi-byte characters in  
 12074 arguments and input files).

12075 **LC\_MESSAGES**

12076 Determine the locale that should be used to affect the format and contents of  
 12077 diagnostic messages written to standard error, and informative messages written  
 12078 to standard output.

12079 **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

12080 **TZ** Determine the timezone in which the time and date are written in the SCCS file. If  
 12081 the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

## 12082 **ASYNCHRONOUS EVENTS**

12083 If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit  
 12084 code. The standard action shall be taken for all other signals; see Section 1.11 (on page 20).

## 12085 **STDOUT**

12086 The standard output shall be used only for the following messages in the POSIX locale:

- 12087 • Prompts (see the *-m* and *-y* options) in the following formats:

12088 "MRs? "

12089 "comments? "

12090 The MR prompt, if written, shall always precede the comments prompt.

- 12091 • A report of each file's activities (unless the *-s* option is specified) in the following format:

12092 "%s\n%d inserted\n%d deleted\n%d unchanged\n", *<New SID>*,  
 12093 *<number of lines inserted>*, *<number of lines deleted>*,  
 12094 *<number of lines unchanged>*

## 12095 **STDERR**

12096 The standard error shall be used only for diagnostic messages.

## 12097 **OUTPUT FILES**

12098 Any SCCS files updated shall be files of an unspecified format.

## 12099 **EXTENDED DESCRIPTION**

### 12100 **System Date and Time**

12101 When a *delta* is added to an SCCS file, the system date and time shall be recorded for the new  
 12102 delta. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the  
 12103 behavior is unspecified.

## 12104 **EXIT STATUS**

12105 The following exit values shall be returned:

12106 0 Successful completion.

12107 >0 An error occurred.

## 12108 **CONSEQUENCES OF ERRORS**

12109 Default.

## 12110 **APPLICATION USAGE**

12111 Problems can arise if the system date and time have been modified (for example, put forward  
 12112 and then back again, or unsynchronized clocks across a network) and can also arise when  
 12113 different values of the *TZ* environment variable are used.

12114 Problems of a similar nature can also arise for the operation of the *get* utility, which records the  
 12115 date and time in the file body.

## 12116 **EXAMPLES**

12117 None.

12118 **RATIONALE**

12119 None.

12120 **FUTURE DIRECTIONS**

12121 None.

12122 **SEE ALSO**12123 Section 1.11 (on page 20), *admin*, *diff*, *get*, *prs*, *rm del*12124 **CHANGE HISTORY**

12125 First released in Issue 2.

12126 **Issue 5**

12127 The output format description in the STDOUT section is corrected.

12128 **Issue 6**

12129 The APPLICATION USAGE section is added.

12130 The normative text is reworded to avoid use of the term “must” for application requirements.

12131 The Open Group Base Resolution bwg2001-007 is applied as follows:

- 12132 • The use of ‘-’ as a file argument is clarified.
- 12133 • The use of STDIN is added.
- 12134 • The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement that
- 12135 implementations re-signal themselves when catching a normally fatal signal.
- 12136 • New text is added to the INPUT FILES section warning that the maximum lines recorded in
- 12137 the file is 99 999.

12138 New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections  
12139 regarding how the system date and time may be taken into account, and the TZ environment  
12140 variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base  
12141 Resolution bwg2001-007.

12142 **NAME**

12143           df — report free disk space

12144 **SYNOPSIS**12145 UP XSI   df [-k] [-P|-t] [*file...*]

12146

12147 **DESCRIPTION**

12148 XSI       The *df* utility shall write the amount of available space and file slots for file systems on which the  
 12149           invoking user has appropriate read access. File systems shall be specified by the *file* operands;  
 12150           when none are specified, information shall be written for all file systems. The format of the  
 12151           default output from *df* is unspecified, but all space figures are reported in 512-byte units, unless  
 12152           the **-k** option is specified. This output shall contain at least the file system names, amount of  
 12153 XSI       available space on each of these file systems, and the number of free file slots, or *inodes*,  
 12154           available; when **-t** is specified, the output shall contain the total allocated space as well.

12155 **OPTIONS**

12156           The *df* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 12157           Utility Syntax Guidelines.

12158           The following options shall be supported:

12159           **-k**           Use 1024-byte units, instead of the default 512-byte units, when writing space  
 12160                           figures.

12161           **-P**           Produce output in the format described in the STDOUT section.

12162 XSI       **-t**           Include total allocated-space figures in the output.

12163 **OPERANDS**

12164           The following operand shall be supported:

12165           *file*           A pathname of a file within the hierarchy of the desired file system. If a file other  
 12166 XSI       than a FIFO, a regular file, a directory, or a special file representing the device  
 12167           containing the file system (for example, **/dev/dsk/0s1**) is specified, the results are  
 12168           unspecified. Otherwise, *df* shall write the amount of free space in the file system  
 12169           containing the specified *file* operand.

12170 **STDIN**

12171           Not used.

12172 **INPUT FILES**

12173           None.

12174 **ENVIRONMENT VARIABLES**

12175           The following environment variables shall affect the execution of *df*:

12176           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
 12177                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 12178                           Internationalization Variables for the precedence of internationalization variables  
 12179                           used to determine the values of locale categories.)

12180           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
 12181                           internationalization variables.

12182           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
 12183                           characters (for example, single-byte as opposed to multi-byte characters in  
 12184                           arguments).

12185 **LC\_MESSAGES**  
 12186 Determine the locale that should be used to affect the format and contents of  
 12187 diagnostic messages written to standard error and informative messages written to  
 12188 standard output.

12189 xSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

12190 **ASYNCHRONOUS EVENTS**  
 12191 Default.

12192 **STDOUT**  
 12193 When both the **-k** and **-P** options are specified, the following header line shall be written (in the  
 12194 POSIX locale):  
 12195 "Filesystem 1024-blocks Used Available Capacity Mounted on\n"  
 12196 When the **-P** option is specified without the **-k** option, the following header line shall be written  
 12197 (in the POSIX locale):  
 12198 "Filesystem 512-blocks Used Available Capacity Mounted on\n"  
 12199 The implementation may adjust the spacing of the header line and the individual data lines so  
 12200 that the information is presented in orderly columns.  
 12201 The remaining output with **-P** shall consist of one line of information for each specified file  
 12202 system. These lines shall be formatted as follows:  
 12203 "%s %d %d %d %d%% %s\n", <file system name>, <total space>,  
 12204 <space used>, <space free>, <percentage used>,  
 12205 <file system root>  
 12206 In the following list, all quantities expressed in 512-byte units (1 024-byte when **-k** is specified)  
 12207 shall be rounded up to the next higher unit. The fields are:  
 12208 <file system name>  
 12209 The name of the file system, in an implementation-defined format.  
 12210 <total space> The total size of the file system in 512-byte units. The exact meaning of this figure  
 12211 is implementation-defined, but should include <space used>, <space free>, plus any  
 12212 space reserved by the system not normally available to a user.  
 12213 <space used> The total amount of space allocated to existing files in the file system, in 512-byte  
 12214 units.  
 12215 <space free> The total amount of space available within the file system for the creation of new  
 12216 files by unprivileged users, in 512-byte units. When this figure is less than or equal  
 12217 to zero, it shall not be possible to create any new files on the file system without  
 12218 first deleting others, unless the process has appropriate privileges. The figure  
 12219 written may be less than zero.  
 12220 <percentage used>  
 12221 The percentage of the normally available space that is currently allocated to all  
 12222 files on the file system. This shall be calculated using the fraction:  
 12223 <space used>/ ( <space used>+ <space free> )  
 12224 expressed as a percentage. This percentage may be greater than 100 if <space free>  
 12225 is less than zero. The percentage value shall be expressed as a positive integer,  
 12226 with any fractional result causing it to be rounded to the next highest integer.



12227 <file system root>

12228 The directory below which the file system hierarchy appears.

12229 xsi The output format is unspecified when **-t** is used.

## 12230 **STDERR**

12231 The standard error shall be used only for diagnostic messages.

## 12232 **OUTPUT FILES**

12233 None.

## 12234 **EXTENDED DESCRIPTION**

12235 None.

## 12236 **EXIT STATUS**

12237 The following exit values shall be returned:

12238 0 Successful completion.

12239 >0 An error occurred.

## 12240 **CONSEQUENCES OF ERRORS**

12241 Default.

## 12242 **APPLICATION USAGE**

12243 On most systems, the “name of the file system, in an implementation-defined format” is the  
12244 special file on which the file system is mounted.

12245 On large file systems, the calculation specified for percentage used can create huge rounding  
12246 errors.

## 12247 **EXAMPLES**

12248 1. The following example writes portable information about the **/usr** file system:

12249 `df -P /usr`

12250 2. Assuming that **/usr/src** is part of the **/usr** file system, the following produces the same  
12251 output as the previous example:

12252 `df -P /usr/src`

## 12253 **RATIONALE**

12254 The behavior of *df* with the **-P** option is the default action of the 4.2 BSD *df* utility. The uppercase  
12255 **-P** was selected to avoid collision with a known industry extension using **-p**.

12256 Historical *df* implementations vary considerably in their default output. It was therefore  
12257 necessary to describe the default output in a loose manner to accommodate all known historical  
12258 implementations and to add a portable option (**-P**) to provide information in a portable format.

12259 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other  
12260 utilities in this volume of IEEE Std 1003.1-2001. This does not mandate that the file system itself  
12261 be based on 512-byte blocks. The **-k** option was added as a compromise measure. It was agreed  
12262 by the standard developers that 512 bytes was the best default unit because of its complete  
12263 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and  
12264 that a **-k** option to switch to 1024-byte units was a good compromise. Users who prefer the  
12265 more logical 1024-byte quantity can easily alias *df* to *df -k* without breaking many historical  
12266 scripts relying on the 512-byte units.

12267 It was suggested that *df* and the various related utilities be modified to access a **BLOCKSIZE**  
12268 environment variable to achieve consistency and user acceptance. Since this is not historical  
12269 practice on any system, it is left as a possible area for system extensions and will be re-evaluated

12270 in a future version if it is widely implemented.

12271 **FUTURE DIRECTIONS**

12272 None.

12273 **SEE ALSO**

12274 *find*

12275 **CHANGE HISTORY**

12276 First released in Issue 2.

12277 **Issue 6**

12278 This utility is marked as part of the User Portability Utilities option.

12279 **NAME**

12280 diff — compare two files

12281 **SYNOPSIS**12282 diff [-c | -e | -f | -C *n*] [-br] *file1 file2*12283 **DESCRIPTION**

12284 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of  
 12285 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be  
 12286 produced if the files are identical.

12287 **OPTIONS**

12288 The *diff* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 12289 12.2, Utility Syntax Guidelines.

12290 The following options shall be supported:

12291 **-b** Cause any amount of white space at the end of a line to be treated as a single  
 12292 <newline> (that is, the white-space characters preceding the <newline> are  
 12293 ignored) and other strings of white-space characters, not including <newline>s, to  
 12294 compare equal.

12295 **-c** Produce output in a form that provides three lines of context.

12296 **-C *n*** Produce output in a form that provides *n* lines of context (where *n* shall be  
 12297 interpreted as a positive decimal integer).

12298 **-e** Produce output in a form suitable as input for the *ed* utility, which can then be  
 12299 used to convert *file1* into *file2*.

12300 **-f** Produce output in an alternative form, similar in format to **-e**, but not intended to  
 12301 be suitable as input for the *ed* utility, and in the opposite order.

12302 **-r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*  
 12303 are both directories.

12304 **OPERANDS**

12305 The following operands shall be supported:

12306 *file1, file2* A pathname of a file to be compared. If either the *file1* or *file2* operand is '-', the  
 12307 standard input shall be used in its place.

12308 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special  
 12309 files, or FIFO special files to any files and shall not compare regular files to directories. Further  
 12310 details are as specified in **Diff Directory Comparison Format** (on page 320). The behavior of *diff*  
 12311 on other file types is implementation-defined when found in directories.

12312 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file  
 12313 contained in the directory file with a filename that is the same as the last component of the non-  
 12314 directory file.

12315 **STDIN**

12316 The standard input shall be used only if one of the *file1* or *file2* operands references standard  
 12317 input. See the INPUT FILES section.

12318 **INPUT FILES**

12319 The input files may be of any type.

12320 **ENVIRONMENT VARIABLES**

12321 The following environment variables shall affect the execution of *diff*:

12322 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 12323 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 12324 Internationalization Variables for the precedence of internationalization variables  
 12325 used to determine the values of locale categories.)

12326 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 12327 internationalization variables.

12328 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 12329 characters (for example, single-byte as opposed to multi-byte characters in  
 12330 arguments and input files).

12331 *LC\_MESSAGES*  
 12332 Determine the locale that should be used to affect the format and contents of  
 12333 diagnostic messages written to standard error and informative messages written to  
 12334 standard output.

12335 *LC\_TIME* Determine the locale for affecting the format of file timestamps written with the  
 12336 *-C* and *-c* options.

12337 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

12338 *TZ* Determine the timezone used for calculating file timestamps written with the *-C*  
 12339 and *-c* options. If *TZ* is unset or null, an unspecified default timezone shall be  
 12340 used.

12341 **ASYNCHRONOUS EVENTS**

12342 Default.

12343 **STDOUT**12344 **Diff Directory Comparison Format**

12345 If both *file1* and *file2* are directories, the following output formats shall be used.

12346 In the POSIX locale, each file that is present in only one directory shall be reported using the  
 12347 following format:

12348 "Only in %s: %s\n", <directory pathname>, <filename>

12349 In the POSIX locale, subdirectories that are common to the two directories may be reported with  
 12350 the following format:

12351 "Common subdirectories: %s and %s\n", <directory1 pathname>,  
 12352 <directory2 pathname>

12353 For each file common to the two directories if the two files are not to be compared, the following  
 12354 format shall be used in the POSIX locale:

12355 "File %s is a %s while file %s is a %s\n", <directory1 pathname>,  
 12356 <file type of directory1 pathname>, <directory2 pathname>,  
 12357 <file type of directory2 pathname>

12358 For each file common to the two directories, if the files are compared and are identical, no output  
 12359 shall be written. If the two files differ, the following format is written:

12360 "diff %s %s %s\n", <diff\_options>, <filename1>, <filename2>

12361 where *<diff\_options>* are the options as specified on the command line.

12362 All directory pathnames listed in this section shall be relative to the original command line  
 12363 arguments. All other names of files listed in this section shall be filenames (pathname  
 12364 components).

12365 **Diff Binary Output Format**

12366 In the POSIX locale, if one or both of the files being compared are not text files, an unspecified  
 12367 format shall be used that contains the pathnames of two files being compared and the string  
 12368 "differ".

12369 If both files being compared are text files, depending on the options specified, one of the  
 12370 following formats shall be used to write the differences.

12371 **Diff Default Output Format**

12372 The default (without **-e**, **-f**, **-c**, or **-C** options) *diff* utility output shall contain lines of these  
 12373 forms:

12374 "%da%d\n", <num1>, <num2>

12375 "%da%d,%d\n", <num1>, <num2>, <num3>

12376 "%dd%d\n", <num1>, <num2>

12377 "%d,%dd%d\n", <num1>, <num2>, <num3>

12378 "%dc%d\n", <num1>, <num2>

12379 "%d,%dc%d\n", <num1>, <num2>, <num3>

12380 "%dc%d,%d\n", <num1>, <num2>, <num3>

12381 "%d,%dc%d,%d\n", <num1>, <num2>, <num3>, <num4>

12382 These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the  
 12383 action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d*  
 12384 and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in  
 12385 *ed*, identical pairs (where *num1*= *num2*) are abbreviated as a single number.

12386 Following each of these lines, *diff* shall write to standard output all lines affected in the first file  
 12387 using the format:

12388 "<Δ%s", <line>

12389 and all lines affected in the second file using the format:

12390 ">Δ%s", <line>

12391 If there are lines affected in both *file1* and *file2* (as with the **c** subcommand), the changes are  
 12392 separated with a line consisting of three hyphens:

12393 "---\n"

**Diff -e Output Format**

With the **-e** option, a script shall be produced that shall, when provided as input to *ed*, along with an appended **w** (write) command, convert *file1* into *file2*. Only the **a** (append), **c** (change), **d** (delete), **i** (insert), and **s** (substitute) commands of *ed* shall be used in this script. Text lines, except those consisting of the single character period ( *'.'* ), shall be output as they appear in the file.

**Diff -f Output Format**

With the **-f** option, an alternative format of script shall be produced. It is similar to that produced by **-e**, with the following differences:

1. It is expressed in reverse sequence; the output of **-e** orders changes from the end of the file to the beginning; the **-f** from beginning to end.
2. The command form *<lines> <command-letter>* used by **-e** is reversed. For example, *10c* with **-e** would be *c10* with **-f**.
3. The form used for ranges of line numbers is *<space>-separated*, rather than comma-separated.

**Diff -c or -C Output Format**

With the **-c** or **-C** option, the output format shall consist of affected lines along with surrounding lines of context. The affected lines shall show which ones need to be deleted or changed in *file1*, and those added from *file2*. With the **-c** option, three lines of context, if available, shall be written before and after the affected lines. With the **-C** option, the user can specify how many lines of context are written. The exact format follows.

The name and last modification time of each file shall be output in the following format:

```
**** %s %s\n", file1, <file1 timestamp>
---- %s %s\n", file2, <file2 timestamp>
```

Each *<file>* field shall be the pathname of the corresponding file being compared. The pathname written for standard input is unspecified.

In the POSIX locale, each *<timestamp>* field shall be equivalent to the output from the following command:

```
date "+%a %b %e %T %Y"
```

without the trailing *<newline>*, executed at the time of last modification of the corresponding file (or the current time, if the file is standard input).

Then, the following output formats shall be applied for every set of changes.

First, a line shall be written in the following format:

```
*****\n"
```

Next, the range of lines in *file1* shall be written in the following format if the range contains two or more lines:

```
**** %d,%d ****\n", <beginning line number>, <ending line number>
```

and the following format otherwise:

```
**** %d ****\n", <ending line number>
```

12433 The ending line number of an empty range shall be the number of the preceding line, or 0 if the  
 12434 range is at the start of the file.

12435 Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected  
 12436 lines shall be written in the following format:

12437 " $\Delta\Delta\%$ s", <unaffected\_line>

12438 Deleted lines shall be written as:

12439 " $-\Delta\%$ s", <deleted\_line>

12440 Changed lines shall be written as:

12441 " $!\Delta\%$ s", <changed\_line>

12442 Next, the range of lines in *file2* shall be written in the following format if the range contains two  
 12443 or more lines:

12444 "--- %d,%d ----\n", <beginning line number>, <ending line number>

12445 and the following format otherwise:

12446 "--- %d ----\n", <ending line number>

12447 Then, lines of context and changed lines shall be written as described in the previous formats.  
 12448 Lines added from *file2* shall be written in the following format:

12449 "+ $\Delta\%$ s", <added\_line>

#### 12450 **STDERR**

12451 The standard error shall be used only for diagnostic messages.

#### 12452 **OUTPUT FILES**

12453 None.

#### 12454 **EXTENDED DESCRIPTION**

12455 None.

#### 12456 **EXIT STATUS**

12457 The following exit values shall be returned:

12458 0 No differences were found.

12459 1 Differences were found.

12460 >1 An error occurred.

#### 12461 **CONSEQUENCES OF ERRORS**

12462 Default.

#### 12463 **APPLICATION USAGE**

12464 If lines at the end of a file are changed and other lines are added, *diff* output may show this as a  
 12465 delete and add, as a change, or as a change and add; *diff* is not expected to know which  
 12466 happened and users should not care about the difference in output as long as it clearly shows the  
 12467 differences between the files.

#### 12468 **EXAMPLES**

12469 If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory  
 12470 named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**,  
 12471 the command:

12472 `diff -r dir1 dir2`

```

12473 could produce output similar to:
12474 Common subdirectories: dir1/x and dir2/x
12475 Only in dir2/x: y
12476 diff -r dir1/x/date.out dir2/x/date.out
12477 1c1
12478 < Mon Jul 2 13:12:16 PDT 1990
12479 ---
12480 > Tue Jun 19 21:41:39 PDT 1990

```

#### 12481 RATIONALE

12482 The **-h** option was omitted because it was insufficiently specified and does not add to  
 12483 applications portability.

12484 Historical implementations employ algorithms that do not always produce a minimum list of  
 12485 differences; the current language about making every effort is the best this volume of  
 12486 IEEE Std 1003.1-2001 can do, as there is no metric that could be employed to judge the quality of  
 12487 implementations against any and all file contents. The statement “This list should be minimal”  
 12488 clearly implies that implementations are not expected to provide the following output when  
 12489 comparing two 100-line files that differ in only one character on a single line:

```

12490 1,100c1,100
12491 all 100 lines from file1 preceded with "< "
12492 ---
12493 all 100 lines from file2 preceded with "> "

```

12494 The “Only in” messages required when the **-r** option is specified are not used by most historical  
 12495 implementations if the **-e** option is also specified. It is required here because it provides useful  
 12496 information that must be provided to update a target directory hierarchy to match a source  
 12497 hierarchy. The “Common subdirectories” messages are written by System V and 4.3 BSD when  
 12498 the **-r** option is specified. They are allowed here but are not required because they are reporting  
 12499 on something that is the same, not reporting a difference, and are not needed to update a target  
 12500 hierarchy.

12501 The **-c** option, which writes output in a format using lines of context, has been included. The  
 12502 format is useful for a variety of reasons, among them being much improved readability and the  
 12503 ability to understand difference changes when the target file has line numbers that differ from  
 12504 another similar, but slightly different, copy. The *patch* utility is most valuable when working  
 12505 with difference listings using the context format. The BSD version of **-c** takes an optional  
 12506 argument specifying the amount of context. Rather than overloading **-c** and breaking the Utility  
 12507 Syntax Guidelines for *diff*, the standard developers decided to add a separate option for  
 12508 specifying a context diff with a specified amount of context (**-C**). Also, the format for context  
 12509 *diffs* was extended slightly in 4.3 BSD to allow multiple changes that are within context lines  
 12510 from each other to be merged together. The output format contains an additional four asterisks  
 12511 after the range of affected lines in the first filename. This was to provide a flag for old programs  
 12512 (like old versions of *patch*) that only understand the old context format. The version of context  
 12513 described here does not require that multiple changes within context lines be merged, but it does  
 12514 not prohibit it either. The extension is upwards-compatible, so any vendors that wish to retain  
 12515 the old version of *diff* can do so by adding the extra four asterisks (that is, utilities that currently  
 12516 use *diff* and understand the new merged format will also understand the old unmerged format,  
 12517 but not *vice versa*).

12518 The substitute command was added as an additional format for the **-e** option. This was added to  
 12519 provide implementations with a way to fix the classic “dot alone on a line” bug present in many  
 12520 versions of *diff*. Since many implementations have fixed this bug, the standard developers  
 12521 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing



12522 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then  
 12523 terminate the append command with a period, and then use the substitute command to convert  
 12524 the two periods into one period.

12525 The BSD-derived *-r* option was added to provide a mechanism for using *diff* to compare two file  
 12526 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not  
 12527 easily reproducible with the *find* utility.

12528 The requirement that *diff* not compare files in some circumstances, even though they have the  
 12529 same name, is based on the actual output of historical implementations. The message specified  
 12530 here is already in use when a directory is being compared to a non-directory. It is extended here  
 12531 to preclude the problems arising from running into FIFOs and other files that would cause *diff* to  
 12532 hang waiting for input with no indication to the user that *diff* was hung. In most common usage,  
 12533 *diff -r* should indicate differences in the file hierarchies, not the difference of contents of devices  
 12534 pointed to by the hierarchies.

12535 Many early implementations of *diff* require seekable files. Since the System Interfaces volume of  
 12536 IEEE Std 1003.1-2001 supports named pipes, the standard developers decided that such a  
 12537 restriction was unreasonable. Note also that the allowed filename – almost always refers to a  
 12538 pipe.

12539 No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in  
 12540 that it prints out all of the differences at the current level, including the statements about all  
 12541 common subdirectories before recursing into those subdirectories.

12542 The message:

12543 "diff %s %s %s\n", <diff\_options>, <filename1>, <filename2>

12544 does not vary by locale because it is the representation of a command, not an English sentence.

#### 12545 **FUTURE DIRECTIONS**

12546 None.

#### 12547 **SEE ALSO**

12548 *cmp, comm, ed, find*

#### 12549 **CHANGE HISTORY**

12550 First released in Issue 2.

#### 12551 **Issue 5**

12552 The FUTURE DIRECTIONS section is added.

#### 12553 **Issue 6**

12554 The following new requirements on POSIX implementations derive from alignment with the  
 12555 Single UNIX Specification:

- 12556 • The *-f* option is added.

12557 The output format for *-c* or *-C* format is changed to align with changes to the IEEE P1003.2b  
 12558 draft standard resulting from IEEE PASC Interpretation 1003.2 #71.

12559 The normative text is reworded to avoid use of the term “must” for application requirements.

12560 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/20 is applied, changing the STDOUT  
 12561 section. This changes the specification of *diff -c* so that it agrees with existing practice when  
 12562 contexts contain zero lines or one line.

12563 **NAME**

12564           dirname — return the directory portion of a pathname

12565 **SYNOPSIS**12566           dirname *string*12567 **DESCRIPTION**

12568           The *string* operand shall be treated as a pathname, as defined in the Base Definitions volume of  
 12569           IEEE Std 1003.1-2001, Section 3.266, Pathname. The string *string* shall be converted to the name  
 12570           of the directory containing the filename corresponding to the last pathname component in  
 12571           *string*, performing actions equivalent to the following steps in order:

- 12572           1. If *string* is //, skip steps 2 to 5.
- 12573           2. If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In  
 12574           this case, skip steps 3 to 8.
- 12575           3. If there are any trailing slash characters in *string*, they shall be removed.
- 12576           4. If there are no slash characters remaining in *string*, *string* shall be set to a single period  
 12577           character. In this case, skip steps 5 to 8.
- 12578           5. If there are any trailing non-slash characters in *string*, they shall be removed.
- 12579           6. If the remaining *string* is //, it is implementation-defined whether steps 7 and 8 are skipped  
 12580           or processed.
- 12581           7. If there are any trailing slash characters in *string*, they shall be removed.
- 12582           8. If the remaining *string* is empty, *string* shall be set to a single slash character.

12583           The resulting string shall be written to standard output.

12584 **OPTIONS**

12585           None.

12586 **OPERANDS**

12587           The following operand shall be supported:

12588           *string*           A string.12589 **STDIN**

12590           Not used.

12591 **INPUT FILES**

12592           None.

12593 **ENVIRONMENT VARIABLES**12594           The following environment variables shall affect the execution of *dirname*:

12595           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 12596           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 12597           Internationalization Variables for the precedence of internationalization variables  
 12598           used to determine the values of locale categories.)

12599           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 12600           internationalization variables.

12601           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 12602           characters (for example, single-byte as opposed to multi-byte characters in  
 12603           arguments).

12604 **LC\_MESSAGES**

12605 Determine the locale that should be used to affect the format and contents of  
 12606 diagnostic messages written to standard error.

12607 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

12608 **ASYNCHRONOUS EVENTS**

12609 Default.

12610 **STDOUT**

12611 The *dirname* utility shall write a line to the standard output in the following format:

12612 "%s\n", <resulting string>

12613 **STDERR**

12614 The standard error shall be used only for diagnostic messages.

12615 **OUTPUT FILES**

12616 None.

12617 **EXTENDED DESCRIPTION**

12618 None.

12619 **EXIT STATUS**

12620 The following exit values shall be returned:

12621 0 Successful completion.

12622 >0 An error occurred.

12623 **CONSEQUENCES OF ERRORS**

12624 Default.

12625 **APPLICATION USAGE**

12626 The definition of *pathname* specifies implementation-defined behavior for pathnames starting  
 12627 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the  
 12628 beginning of a pathname unless they can ensure that there are more or less than two or are  
 12629 prepared to deal with the implementation-defined consequences.

12630 **EXAMPLES**

| Command                 | Results     |
|-------------------------|-------------|
| <i>dirname</i> /        | /           |
| <i>dirname</i> //       | / or //     |
| <i>dirname</i> /a/b/    | /a          |
| <i>dirname</i> //a//b// | //a         |
| <i>dirname</i>          | Unspecified |
| <i>dirname</i> a        | .( \$? = 0) |
| <i>dirname</i> ""       | .( \$? = 0) |
| <i>dirname</i> /a       | /           |
| <i>dirname</i> /a/b     | /a          |
| <i>dirname</i> a/b      | a           |

12642 **RATIONALE**

12643 The *dirname* utility originated in System III. It has evolved through the System V releases to a  
 12644 version that matches the requirements specified in this description in System V Release 3. 4.3  
 12645 BSD and earlier versions did not include *dirname*.

12646 The behaviors of *basename* and *dirname* in this volume of IEEE Std 1003.1-2001 have been  
 12647 coordinated so that when *string* is a valid pathname:

12648           \$(basename "*string*")

12649           would be a valid filename for the file in the directory:

12650           \$(dirname "*string*")

12651           This would not work for the versions of these utilities in early proposals due to the way  
12652           processing of trailing slashes was specified. Consideration was given to leaving processing  
12653           unspecified if there were trailing slashes, but this cannot be done; the Base Definitions volume of  
12654           IEEE Std 1003.1-2001, Section 3.266, Pathname allows trailing slashes. The *basename* and *dirname*  
12655           utilities have to specify consistent handling for all valid pathnames.

12656 **FUTURE DIRECTIONS**

12657           None.

12658 **SEE ALSO**

12659           *basename*, Section 2.5 (on page 33)

12660 **CHANGE HISTORY**

12661           First released in Issue 2.

12662 **NAME**

12663           du — estimate file space usage

12664 **SYNOPSIS**12665 UP       du [-a | -s] [-kx] [-H | -L] [*file* ...]

12666

12667 **DESCRIPTION**

12668       By default, the *du* utility shall write to standard output the size of the file space allocated to, and  
 12669       the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the  
 12670       specified files. By default, when a symbolic link is encountered on the command line or in the  
 12671       file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the  
 12672       link), and shall not follow the link to another portion of the file hierarchy. The size of the file  
 12673       space allocated to a file of type directory shall be defined as the sum total of space allocated to  
 12674       all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

12675       When *du* cannot *stat()* files or *stat()* or read directories, it shall report an error condition and the  
 12676       final exit status is affected. Files with multiple links shall be counted and written for only one  
 12677       entry. The directory entry that is selected in the report is unspecified. By default, file sizes shall  
 12678       be written in 512-byte units, rounded up to the next 512-byte unit.

12679 **OPTIONS**

12680       The *du* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 12681       Utility Syntax Guidelines.

12682       The following options shall be supported:

12683       **-a**           In addition to the default output, report the size of each file not of type directory in  
 12684       the file hierarchy rooted in the specified file. Regardless of the presence of the **-a**  
 12685       option, non-directories given as *file* operands shall always be listed.

12686       **-H**           If a symbolic link is specified on the command line, *du* shall count the size of the  
 12687       file or file hierarchy referenced by the link.

12688       **-k**           Write the files sizes in units of 1 024 bytes, rather than the default 512-byte units.

12689       **-L**           If a symbolic link is specified on the command line or encountered during the  
 12690       traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy  
 12691       referenced by the link.

12692       **-s**           Instead of the default output, report only the total sum for each of the specified  
 12693       files.

12694       **-x**           When evaluating file sizes, evaluate only those files that have the same device as  
 12695       the file specified by the *file* operand.

12696       Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
 12697       an error. The last option specified shall determine the behavior of the utility.

12698 **OPERANDS**

12699       The following operand shall be supported:

12700       *file*        The pathname of a file whose size is to be written. If no *file* is specified, the current  
 12701       directory shall be used.

12702 **STDIN**

12703       Not used.

12704 **INPUT FILES**

12705       None.

12706 **ENVIRONMENT VARIABLES**12707       The following environment variables shall affect the execution of *du*:

12708       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 12709                   (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 12710                   Internationalization Variables for the precedence of internationalization variables  
 12711                   used to determine the values of locale categories.)

12712       *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 12713                   internationalization variables.

12714       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 12715                   characters (for example, single-byte as opposed to multi-byte characters in  
 12716                   arguments).

12717       *LC\_MESSAGES*  
 12718                   Determine the locale that should be used to affect the format and contents of  
 12719                   diagnostic messages written to standard error.

12720 xsi       *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

12721 **ASYNCHRONOUS EVENTS**

12722       Default.

12723 **STDOUT**

12724       The output from *du* shall consist of the amount of space allocated to a file and the name of the  
 12725       file, in the following format:

12726       "%d %s\n", <size>, <pathname>

12727 **STDERR**

12728       The standard error shall be used only for diagnostic messages.

12729 **OUTPUT FILES**

12730       None.

12731 **EXTENDED DESCRIPTION**

12732       None.

12733 **EXIT STATUS**

12734       The following exit values shall be returned:

12735       0   Successful completion.

12736       >0   An error occurred.

12737 **CONSEQUENCES OF ERRORS**

12738       Default.

12739 **APPLICATION USAGE**

12740 None.

12741 **EXAMPLES**

12742 None.

12743 **RATIONALE**

12744 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other  
12745 utilities in this volume of IEEE Std 1003.1-2001. This does not mandate that the file system itself  
12746 be based on 512-byte blocks. The **-k** option was added as a compromise measure. It was agreed  
12747 by the standard developers that 512 bytes was the best default unit because of its complete  
12748 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and  
12749 that a **-k** option to switch to 1024-byte units was a good compromise. Users who prefer the  
12750 1024-byte quantity can easily alias *du* to *du -k* without breaking the many historical scripts  
12751 relying on the 512-byte units.

12752 The **-b** option was added to an early proposal to provide a resolution to the situation where  
12753 System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-  
12754 defined concept. (In common usage, the block size is 512 bytes for System V and 1024 bytes for  
12755 BSD systems.) However, **-b** was later deleted, since the default was eventually decided as 512-  
12756 byte units.

12757 Historical file systems provided no way to obtain exact figures for the space allocation given to  
12758 files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks*  
12759 being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is  
12760 space used by the file system in the storage of the file, but that need not be counted in the space  
12761 allocated to the file. A *sparse* file is one in which an *lseek()* call has been made to a position  
12762 beyond the end of the file and data has subsequently been written at that point. A file system  
12763 need not allocate all the intervening zero-filled blocks to such a file. It is up to the  
12764 implementation to define exactly how accurate its methods are.

12765 The **-a** and **-s** options were mutually-exclusive in the original version of *du*. The POSIX Shell  
12766 and Utilities description is implied by the language in the SVID where **-s** is described as causing  
12767 “only the grand total” to be reported. Some systems may produce output for **-sa**, but a Strictly  
12768 Conforming POSIX Shell and Utilities Application cannot use that combination.

12769 The **-a** and **-s** options were adopted from the SVID except that the System V behavior of not  
12770 listing non-directories explicitly given as operands, unless the **-a** option is specified, was  
12771 considered a bug; the BSD-based behavior (report for all operands) is mandated. The default  
12772 behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no  
12773 messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and  
12774 Utilities default behavior shall be to produce such messages. These messages can be turned off  
12775 with shell redirection to achieve the System V behavior.

12776 The **-x** option is historical practice on recent BSD systems. It has been adopted by this volume of  
12777 IEEE Std 1003.1-2001 because there was no other historical method of limiting the *du* search to a  
12778 single file hierarchy. This limitation of the search is necessary to make it possible to obtain file  
12779 space usage information about a file system on which other file systems are mounted, without  
12780 having to resort to a lengthy *find* and *awk* script.

12781 **FUTURE DIRECTIONS**

12782 None.

12783 **SEE ALSO**

12784 *ls*, the System Interfaces volume of IEEE Std 1003.1-2001, *stat()*

12785 **CHANGE HISTORY**

12786 First released in Issue 2.

12787 **Issue 6**

12788 This utility is marked as part of the User Portability Utilities option.

12789 The APPLICATION USAGE section is added.

12790 The obsolescent **-r** option has been removed.

12791 The Open Group Corrigendum U025/3 is applied. The *du* utility is reinstated, as it had  
12792 incorrectly been marked LEGACY in Issue 5.

12793 The **-H** and **-L** options for symbolic links are added as described in the IEEE P1003.2b draft  
12794 standard.



12795 **NAME**

12796       echo — write arguments to standard output

12797 **SYNOPSIS**12798       echo [*string* ...]12799 **DESCRIPTION**12800       The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are  
12801       no arguments, only the <newline> is written.12802 **OPTIONS**12803       The *echo* utility shall not recognize the "--" argument in the manner specified by Guideline 10  
12804       of the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines;  
12805       "--" shall be recognized as a string operand.

12806       Implementations shall not support any options.

12807 **OPERANDS**

12808       The following operands shall be supported:

12809       *string*       A string to be written to standard output. If the first operand is **-n**, or if any of the  
12810       operands contain a backslash ('\\') character, the results are implementation-  
12811       defined.12812 XSI               On XSI-conformant systems, if the first operand is **-n**, it shall be treated as a string,  
12813                       not an option. The following character sequences shall be recognized on XSI-  
12814                       conformant systems within any of the arguments:

12815                       \\a       Write an &lt;alert&gt;.

12816                       \\b       Write a &lt;backspace&gt;.

12817                       \\c       Suppress the <newline> that otherwise follows the final argument in the  
12818                       output. All characters following the '\\c' in the arguments shall be  
12819                       ignored.

12820                       \\f       Write a &lt;form-feed&gt;.

12821                       \\n       Write a &lt;newline&gt;.

12822                       \\r       Write a &lt;carriage-return&gt;.

12823                       \\t       Write a &lt;tab&gt;.

12824                       \\v       Write a &lt;vertical-tab&gt;.

12825                       \\       Write a backslash character.

12826                       \\0*num*   Write an 8-bit value that is the zero, one, two, or three-digit octal number  
12827                       *num*.

12828

12829 **STDIN**

12830       Not used.

12831 **INPUT FILES**

12832       None.

12833 **ENVIRONMENT VARIABLES**12834       The following environment variables shall affect the execution of *echo*:12835       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
12836       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,

12837 Internationalization Variables for the precedence of internationalization variables  
 12838 used to determine the values of locale categories.)

12839 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 12840 internationalization variables.

12841 XSI **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 12842 characters (for example, single-byte as opposed to multi-byte characters in  
 12843 arguments).

12844 **LC\_MESSAGES**  
 12845 Determine the locale that should be used to affect the format and contents of  
 12846 diagnostic messages written to standard error.

12847 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

## 12848 ASYNCHRONOUS EVENTS

12849 Default.

## 12850 STDOUT

12851 The *echo* utility arguments shall be separated by single <space>s and a <newline> shall follow  
 12852 XSI the last argument. Output transformations shall occur based on the escape sequences in the  
 12853 input. See the OPERANDS section.

## 12854 STDERR

12855 The standard error shall be used only for diagnostic messages.

## 12856 OUTPUT FILES

12857 None.

## 12858 EXTENDED DESCRIPTION

12859 None.

## 12860 EXIT STATUS

12861 The following exit values shall be returned:

12862 0 Successful completion.

12863 >0 An error occurred.

## 12864 CONSEQUENCES OF ERRORS

12865 Default.

## 12866 APPLICATION USAGE

12867 It is not possible to use *echo* portably across all POSIX systems unless both **-n** (as the first  
 12868 argument) and escape sequences are omitted.

12869 The *printf* utility can be used portably to emulate any of the traditional behaviors of the *echo*  
 12870 utility as follows (assuming that *IFS* has its standard value or is unset):

12871 • The historic System V *echo* and the requirements on XSI implementations in this volume of  
 12872 IEEE Std 1003.1-2001 are equivalent to:

12873 `printf "%b\n" "$@"`

12874 • The BSD *echo* is equivalent to:

12875 `if [ "X$1" = "X-n" ]`

12876 `then`

12877 `shift`

12878 `printf "%s" "$@"`

12879 `else`

```
12880 printf "%s\n" "$*"
12881 fi
```

12882 New applications are encouraged to use *printf* instead of *echo*.

#### 12883 **EXAMPLES**

12884 None.

#### 12885 **RATIONALE**

12886 The *echo* utility has not been made obsolescent because of its extremely widespread use in  
 12887 historical applications. Conforming applications that wish to do prompting without <newline>s  
 12888 or that could possibly be expecting to echo a **-n**, should use the *printf* utility derived from the  
 12889 Ninth Edition system.

12890 As specified, *echo* writes its arguments in the simplest of ways. The two different historical  
 12891 versions of *echo* vary in fatally incompatible ways.

12892 The BSD *echo* checks the first argument for the string **-n** which causes it to suppress the  
 12893 <newline> that would otherwise follow the final argument in the output.

12894 The System V *echo* does not support any options, but allows escape sequences within its  
 12895 operands, as described for XSI implementations in the OPERANDS section.

12896 The *echo* utility does not support Utility Syntax Guideline 10 because historical applications  
 12897 depend on *echo* to echo *all* of its arguments, except for the **-n** option in the BSD version.

#### 12898 **FUTURE DIRECTIONS**

12899 None.

#### 12900 **SEE ALSO**

12901 *printf*

#### 12902 **CHANGE HISTORY**

12903 First released in Issue 2.

#### 12904 **Issue 5**

12905 In the OPTIONS section, the last sentence is changed to indicate that implementations “do not”  
 12906 support any options; in the previous issue this said “need not”.

#### 12907 **Issue 6**

12908 The following new requirements on POSIX implementations derive from alignment with the  
 12909 Single UNIX Specification:

- 12910 • A set of character sequences is defined as *string* operands.
- 12911 • *LC\_CTYPE* is added to the list of environment variables affecting *echo*.
- 12912 • In the OPTIONS section, implementations shall not support any options.

12913 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/21 is applied, so that the *echo* utility can  
 12914 accommodate historical BSD behavior.

12915 **NAME**

12916       ed — edit text

12917 **SYNOPSIS**12918       ed [-p *string*] [-s] [*file*]12919 **DESCRIPTION**

12920       The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*.  
 12921       In command mode the input characters shall be interpreted as commands, and in input mode  
 12922       they shall be interpreted as text. See the EXTENDED DESCRIPTION section.

12923 **OPTIONS**

12924       The *ed* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 12925       Utility Syntax Guidelines.

12926       The following options shall be supported:

12927       -p *string*     Use *string* as the prompt string when in command mode. By default, there shall be  
 12928       no prompt string.

12929       -s             Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the '!'  
 12930       prompt after a !*command*.

12931 **OPERANDS**

12932       The following operand shall be supported:

12933       *file*           If the *file* argument is given, *ed* shall simulate an **e** command on the file named by  
 12934       the pathname, *file*, before accepting commands from the standard input. If the *file*  
 12935       operand is '-', the results are unspecified.

12936 **STDIN**

12937       The standard input shall be a text file consisting of commands, as described in the EXTENDED  
 12938       DESCRIPTION section.

12939 **INPUT FILES**

12940       The input files shall be text files.

12941 **ENVIRONMENT VARIABLES**

12942       The following environment variables shall affect the execution of *ed*:

12943       HOME           Determine the pathname of the user's home directory.

12944       LANG           Provide a default value for the internationalization variables that are unset or null.  
 12945       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 12946       Internationalization Variables for the precedence of internationalization variables  
 12947       used to determine the values of locale categories.)

12948       LC\_ALL          If set to a non-empty string value, override the values of all the other  
 12949       internationalization variables.

12950       LC\_COLLATE     Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 12951       character collating elements within regular expressions.  
 12952

12953       LC\_CTYPE       Determine the locale for the interpretation of sequences of bytes of text data as  
 12954       characters (for example, single-byte as opposed to multi-byte characters in  
 12955       arguments and input files) and the behavior of character classes within regular  
 12956       expressions.

12957       LC\_MESSAGES    Determine the locale that should be used to affect the format and contents of  
 12958

12959 diagnostic messages written to standard error and informative messages written to  
12960 standard output.

12961 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 12962 ASYNCHRONOUS EVENTS

12963 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS  
12964 section in Section 1.11 (on page 20)) with the following exceptions:

12965 SIGINT The *ed* utility shall interrupt its current activity, write the string "?\n" to standard  
12966 output, and return to command mode (see the EXTENDED DESCRIPTION  
12967 section).

12968 SIGHUP If the buffer is not empty and has changed since the last write, the *ed* utility shall  
12969 attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the  
12970 current directory shall be used; if that fails, the file named **ed.hup** in the directory  
12971 named by the *HOME* environment variable shall be used. In any case, the *ed* utility  
12972 shall exit without returning to command mode.

12973 SIGQUIT The *ed* utility shall ignore this event.

## 12974 STDOUT

12975 Various editing commands and the prompting feature (see **-p**) write to standard output, as  
12976 described in the EXTENDED DESCRIPTION section.

## 12977 STDERR

12978 The standard error shall be used only for diagnostic messages.

## 12979 OUTPUT FILES

12980 The output files shall be text files whose formats are dependent on the editing commands given.

## 12981 EXTENDED DESCRIPTION

12982 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have  
12983 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.

12984 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a  
12985 single-character *command*, possibly followed by parameters to that command. These addresses  
12986 specify one or more lines in the buffer. Every command that requires addresses has default  
12987 addresses, so that the addresses very often can be omitted. If the **-p** option is specified, the  
12988 prompt string shall be written to standard output before each command is read.

12989 In general, only one command can appear on a line. Certain commands allow text to be input.  
12990 This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be  
12991 in *input mode*. In this mode, no commands shall be recognized; all input is merely collected.  
12992 Input mode is terminated by entering a line consisting of two characters: a period ( **.** )  
12993 followed by a <newline>. This line is not considered part of the input text.

## 12994 Regular Expressions in ed

12995 The *ed* utility shall support basic regular expressions, as described in the Base Definitions  
12996 volume of IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions. Since regular  
12997 expressions in *ed* are always matched against single lines (excluding the terminating  
12998 <newline>s), never against any larger section of text, there is no way for a regular expression to  
12999 match a <newline>.

13000 A null RE shall be equivalent to the last RE encountered.

13001 Regular expressions are used in addresses to specify lines, and in some commands (for example,  
13002 the **s** substitute command) to specify portions of a line to be substituted.

## 13003 **Addresses in ed**

13004 Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a  
 13005 command. The current line number is the address of the current line. If the edit buffer is not  
 13006 empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

13007 Addresses shall be constructed as follows:

- 13008 1. The period character ( ' . ' ) shall address the current line.
- 13009 2. The dollar sign character ( ' \$ ' ) shall address the last line of the edit buffer.
- 13010 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 13011 4. The apostrophe-x character pair ( " ' x " ) shall address the line marked with the mark name  
 13012 character *x*, which shall be a lowercase letter from the portable character set. It shall be an  
 13013 error if the character has not been set to mark a line or if the line that was marked is not  
 13014 currently present in the edit buffer.
- 13015 5. A BRE enclosed by slash characters ( ' / ' ) shall address the first line found by searching  
 13016 forwards from the line following the current line toward the end of the edit buffer and  
 13017 stopping at the first line for which the line excluding the terminating <newline> matches  
 13018 the BRE. The BRE consisting of a null BRE delimited by a pair of slash characters shall  
 13019 address the next line for which the line excluding the terminating <newline> matches the  
 13020 last BRE encountered. In addition, the second slash can be omitted at the end of a  
 13021 command line. Within the BRE, a backslash-slash pair ( " \ / " ) shall represent a literal slash  
 13022 instead of the BRE delimiter. If necessary, the search shall wrap around to the beginning of  
 13023 the buffer and continue up to and including the current line, so that the entire buffer is  
 13024 searched.
- 13025 6. A BRE enclosed by question-mark characters ( ' ? ' ) shall address the first line found by  
 13026 searching backwards from the line preceding the current line toward the beginning of the  
 13027 edit buffer and stopping at the first line for which the line excluding the terminating  
 13028 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of  
 13029 question-mark characters ( " ? ? " ) shall address the previous line for which the line  
 13030 excluding the terminating <newline> matches the last BRE encountered. In addition, the  
 13031 second question-mark can be omitted at the end of a command line. Within the BRE, a  
 13032 backslash-question-mark pair ( " \ ? " ) shall represent a literal question mark instead of the  
 13033 BRE delimiter. If necessary, the search shall wrap around to the end of the buffer and  
 13034 continue up to and including the current line, so that the entire buffer is searched.
- 13035 7. A plus-sign ( ' + ' ) or hyphen character ( ' - ' ) followed by a decimal number shall address  
 13036 the current line plus or minus the number. A plus-sign or hyphen character not followed  
 13037 by a decimal number shall address the current line plus or minus 1.

13038 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.  
 13039 Address offsets are constructed as follows:

- 13040 • A plus-sign or hyphen character followed by a decimal number shall add or subtract,  
 13041 respectively, the indicated number of lines to or from the address. A plus-sign or hyphen  
 13042 character not followed by a decimal number shall add or subtract 1 to or from the address.
- 13043 • A decimal number shall add the indicated number of lines to the address.

13044 It shall not be an error for an intermediate address value to be less than zero or greater than the  
 13045 last line in the edit buffer. It shall be an error for the final address value to be less than zero or  
 13046 greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a  
 13047 matching line.

13048 Commands accept zero, one, or two addresses. If more than the required number of addresses  
 13049 are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more  
 13050 than the required number of addresses are provided to a command, the addresses specified first  
 13051 shall be evaluated and then discarded until the maximum number of valid addresses remain, for  
 13052 the specified command.

13053 Addresses shall be separated from each other by a comma (',' ) or semicolon character (';' ).  
 13054 In the case of a semicolon separator, the current line ('.' ) shall be set to the first address, and  
 13055 only then will the second address be calculated. This feature can be used to determine the  
 13056 starting line for forwards and backwards searches; see rules 5. and 6.

13057 Addresses can be omitted on either side of the comma or semicolon separator, in which case the  
 13058 resulting address pairs shall be as follows:

| Specified | Resulting   |
|-----------|-------------|
| ,         | 1 , \$      |
| , addr    | 1 , addr    |
| addr ,    | addr , addr |
| ;         | . ; \$      |
| ; addr    | . ; addr    |
| addr ;    | addr ; addr |

13066 Any <blank>s included between addresses, address separators, or address offsets shall be  
 13067 ignored.

#### 13068 Commands in ed

13069 In the following list of *ed* commands, the default addresses are shown in parentheses. The  
 13070 number of addresses shown in the default shall be the number expected by the command. The  
 13071 parentheses are not part of the address; they show that the given addresses are the default.

13072 It is generally invalid for more than one command to appear on a line. However, any command  
 13073 (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for  
 13074 the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be  
 13075 written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used  
 13076 with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but  
 13077 it is unspecified whether the suffix writes the current line again in the requested format or  
 13078 whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l**  
 13079 suffix) shall either write just the current line or write it twice—once as specified for **p** and once  
 13080 as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

13081 Each address component can be preceded by zero or more <blank>s. The command letter can be  
 13082 preceded by zero or more <blank>s. If a suffix letter (**l**, **n**, or **p**) is given, the application shall  
 13083 ensure that it immediately follows the command.

13084 The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the  
 13085 command letter by one or more <blank>s.

13086 If changes have been made in the buffer since the last **w** command that wrote the entire buffer,  
 13087 *ed* shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.  
 13088 The *ed* utility shall write the string:

13089 "?\n"

13090 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to  
 13091 standard output and shall continue in command mode with the current line number unchanged.  
 13092 If the **e** or **q** command is repeated with no intervening command, it shall take effect.

13093 If a terminal disconnect is detected:

13094 • If the buffer is not empty and has changed since the last write, the *ed* utility shall attempt to  
13095 write a copy of the buffer to a file named **ed.hup** in the current directory. If this write fails, *ed*  
13096 shall attempt to write a copy of the buffer to a filename **ed.hup** in the directory named by the  
13097 *HOME* environment variable. If both these attempts fail, *ed* shall exit without saving the  
13098 buffer.

13099 • The *ed* utility shall not write the file to the currently remembered pathname or return to  
13100 command mode, and shall terminate with a non-zero exit status.

13101 If an end-of-file is detected on standard input:

13102 • If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command mode.  
13103 It is unspecified if any partially entered lines (that is, input text without a terminating  
13104 <newline>) are discarded from the input text.

13105 • If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

13106 If the closing delimiter of an RE or of a replacement string (for example, ' / ') in a **g**, **G**, **s**, **v**, or **V**  
13107 command would be the last character before a <newline>, that delimiter can be omitted, in  
13108 which case the addressed line shall be written. For example, the following pairs of commands  
13109 are equivalent:

```
13110 s/s1/s2 s/s1/s2/p
13111 g/s1 g/s1/p
13112 ?s1 ?s1?
```

13113 If an invalid command is entered, *ed* shall write the string:

13114 "?\n"

13115 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to  
13116 standard output and shall continue in command mode with the current line number unchanged.

## 13117 Append Command

13118 Synopsis:     ( . ) a  
13119                 <text>  
13120                 .

13121 The **a** command shall read the given text and append it after the addressed line; the current line  
13122 number shall become the address of the last inserted line or, if there were none, the addressed  
13123 line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at  
13124 the beginning of the buffer.

## 13125 Change Command

13126 Synopsis:     ( . , . ) c  
13127                 <text>  
13128                 .

13129 The **c** command shall delete the addressed lines, then accept input text that replaces these lines;  
13130 the current line shall be set to the address of the last line input; or, if there were none, at the line  
13131 after the last line deleted; if the lines deleted were originally at the end of the buffer, the current  
13132 line number shall be set to the address of the new last line; if no lines remain in the buffer, the  
13133 current line number shall be set to zero. Address 0 shall be valid for this command; it shall be  
13134 interpreted as if address 1 were specified.



**Delete Command**

Synopsis:     (.,.)d

The **d** command shall delete the addressed lines from the buffer. The address of the line after the last line deleted shall become the current line number; if the lines deleted were originally at the end of the buffer, the current line number shall be set to the address of the new last line; if no lines remain in the buffer, the current line number shall be set to zero.

**Edit Command**

Synopsis:     e [*file*]

The **e** command shall delete the entire contents of the buffer and then read in the file named by the pathname *file*. The current line number shall be set to the address of the last line of the buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see the **f** command). The number of bytes read shall be written to standard output, unless the **—s** option was specified, in the following format:

"%d\n", <number of bytes read>

The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**, and **w** commands. If *file* is replaced by '!', the rest of the line shall be taken to be a shell command line whose output is to be read. Such a shell command line shall not be remembered as the current *file*. All marks shall be discarded upon the completion of a successful **e** command. If the buffer has changed since the last time the entire buffer was written, the user shall be warned, as described previously.

**Edit Without Checking Command**

Synopsis:     E [*file*]

The **E** command shall possess all properties and restrictions of the **e** command except that the editor shall not check to see whether any changes have been made to the buffer since the last **w** command.

**Filename Command**

Synopsis:     f [*file*]

If *file* is given, the **f** command shall change the currently remembered pathname to *file*; whether the name is changed or not, it shall then write the (possibly new) currently remembered pathname to the standard output in the following format:

"%s\n", <pathname>

The current line number shall be unchanged.

**Global Command**

Synopsis:     (1,\$)g/RE/command list

In the **g** command, the first step shall be to mark every line for which the line excluding the terminating <newline> matches the given RE. Then, going sequentially from the beginning of the file to the end of the file, the given *command list* shall be executed for each marked line, with the current line number set to the address of that line. Any line modified by the *command list* shall be unmarked. When the **g** command completes, the current line number shall have the value assigned by the last command in the *command list*. If there were no matching lines, the current line number shall not be changed. A single command or the first of a list of commands

shall appear on the same line as the global command. All lines of a multi-line list except the last line shall be ended with a backslash preceding the terminating <newline>; the **a**, **i**, and **c** commands and associated input are permitted. The ' . ' terminating input mode can be omitted if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p** command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined results. Any character other than <space> or <newline> can be used instead of a slash to delimit the RE. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by a backslash.

### Interactive Global Command

*Synopsis:* (1, \$)G/RE/

In the **G** command, the first step shall be to mark every line for which the line excluding the terminating <newline> matches the given RE. Then, for every such line, that line shall be written, the current line number shall be set to the address of that line, and any one command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline> shall act as a null command (causing no action to be taken on the current line); an '&' shall cause the re-execution of the most recent non-null command executed within the current invocation of **G**. Note that the commands input as part of the execution of the **G** command can address and affect any lines in the buffer. Any line modified by the command shall be unmarked. The final value of the current line number shall be the value set by the last command successfully executed. (Note that the last command successfully executed shall be the **G** command itself if a command fails or the null command is specified.) If there were no matching lines, the current line number shall not be changed. The **G** command can be terminated by a SIGINT signal. Any character other than <space> or <newline> can be used instead of a slash to delimit the RE and the replacement. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by a backslash.

### Help Command

*Synopsis:* h

The **h** command shall write a short message to standard output that explains the reason for the most recent '?' notification. The current line number shall be unchanged.

### Help-Mode Command

*Synopsis:* H

The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command) shall be written to standard output for all subsequent '?' notifications. The **H** command alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on, the **H** command also explains the previous '?' notification, if there was one. The current line number shall be unchanged.

### Insert Command

*Synopsis:* (.)i  
<text>  
.

The **i** command shall insert the given text before the addressed line; the current line is set to the last inserted line or, if there was none, to the addressed line. This command differs from the **a** command only in the placement of the input text. Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified.

13220 **Join Command**13221 *Synopsis:* ( . , .+1 ) j

13222 The **j** command shall join contiguous lines by removing the appropriate <newline>s. If exactly  
 13223 one address is given, this command shall do nothing. If lines are joined, the current line number  
 13224 shall be set to the address of the joined line; otherwise, the current line number shall be  
 13225 unchanged.

13226 **Mark Command**13227 *Synopsis:* ( . ) kx

13228 The **k** command shall mark the addressed line with name x, which the application shall ensure is  
 13229 a lowercase letter from the portable character set. The address "x" shall then refer to this line;  
 13230 the current line number shall be unchanged.

13231 **List Command**13232 *Synopsis:* ( . , . ) l

13233 The **l** command shall write to standard output the addressed lines in a visually unambiguous  
 13234 form. The characters listed in the Base Definitions volume of IEEE Std 1003.1-2001, Table 5-1,  
 13235 Escape Sequences and Associated Actions ('\\', '\a', '\b', '\f', '\r', '\t', '\v') shall  
 13236 be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-  
 13237 printable characters not in the table shall be written as one three-digit octal number (with a  
 13238 preceding backslash character) for each byte in the character (most significant byte first). If the  
 13239 size of a byte on the system is greater than nine bits, the format used for non-printable characters  
 13240 is implementation-defined.

13241 Long lines shall be folded, with the point of folding indicated by <newline> preceded by a  
 13242 backslash; the length at which folding occurs is unspecified, but should be appropriate for the  
 13243 output device. The end of each line shall be marked with a '\$', and '\$' characters within the  
 13244 text shall be written with a preceding backslash. An **l** command can be appended to any other  
 13245 command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the address of  
 13246 the last line written.

13247 **Move Command**13248 *Synopsis:* ( . , . ) maddress

13249 The **m** command shall reposition the addressed lines after the line addressed by *address*.  
 13250 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning  
 13251 of the buffer. It shall be an error if address *address* falls within the range of moved lines. The  
 13252 current line number shall be set to the address of the last line moved.

13253 **Number Command**13254 *Synopsis:* ( . , . ) n

13255 The **n** command shall write to standard output the addressed lines, preceding each line by its  
 13256 line number and a <tab>; the current line number shall be set to the address of the last line  
 13257 written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

**Print Command**

Synopsis: ( , . ) p

The **p** command shall write to standard output the addressed lines; the current line number shall be set to the address of the last line written. The **p** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

**Prompt Command**

Synopsis: P

The **P** command shall cause *ed* to prompt with an asterisk ( **' \* '** ) (or *string*, if **-p** is specified) for all subsequent commands. The **P** command alternatively shall turn this mode on and off; it shall be initially on if the **-p** option is specified; otherwise, off. The current line number shall be unchanged.

**Quit Command**

Synopsis: q

The **q** command shall cause *ed* to exit. If the buffer has changed since the last time the entire buffer was written, the user shall be warned, as described previously.

**Quit Without Checking Command**

Synopsis: Q

The **Q** command shall cause *ed* to exit without checking whether changes have been made in the buffer since the last **w** command.

**Read Command**

Synopsis: ( \$ ) r [ *file* ]

The **r** command shall read in the file named by the pathname *file* and append it after the addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be used (see the **e** and **f** commands). The currently remembered pathname shall not be changed unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file to be read at the beginning of the buffer. If the read is successful, and **-s** was not specified, the number of bytes read shall be written to standard output in the following format:

```
"%d\n", <number of bytes read>
```

The current line number shall be set to the address of the last line read in. If *file* is replaced by **' ! '**, the rest of the line shall be taken to be a shell command line whose output is to be read. Such a shell command line shall not be remembered as the current pathname.

**Substitute Command**

Synopsis: ( , . ) s / *RE* / *replacement* / *flags*

The **s** command shall search each addressed line for an occurrence of the specified *RE* and replace either the first or all (non-overlapped) matched strings with the *replacement*; see the following description of the **g** suffix. It is an error if the substitution fails on every addressed line. Any character other than *<space>* or *<newline>* can be used instead of a slash to delimit the *RE* and the replacement. Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is preceded by a backslash. The current line shall be set to the address of the last line on which a substitution occurred.

13298 An ampersand ('&') appearing in the *replacement* shall be replaced by the string matching the  
 13299 RE on the current line. The special meaning of '&' in this context can be suppressed by  
 13300 preceding it by backslash. As a more general feature, the characters '\n', where *n* is a digit,  
 13301 shall be replaced by the text matched by the corresponding back-reference expression. When the  
 13302 character '%' is the only character in the *replacement*, the *replacement* used in the most recent  
 13303 substitute command shall be used as the *replacement* in the current substitute command; if there  
 13304 was no previous substitute command, the use of '%' in this manner shall be an error. The '%'   
 13305 shall lose its special meaning when it is in a replacement string of more than one character or is  
 13306 preceded by a backslash. For each backslash ('\') encountered in scanning *replacement* from  
 13307 beginning to end, the following character shall lose its special meaning (if any). It is unspecified  
 13308 what special meaning is given to any character other than '&', '\', '%', or digits.

13309 A line can be split by substituting a <newline> into it. The application shall ensure it escapes the  
 13310 <newline> in the *replacement* by preceding it by backslash. Such substitution cannot be done as  
 13311 part of a **g** or **v** command list. The current line number shall be set to the address of the last line  
 13312 on which a substitution is performed. If no substitution is performed, the current line number  
 13313 shall be unchanged. If a line is split, a substitution shall be considered to have been performed  
 13314 on each of the new lines for the purpose of determining the new current line number. A  
 13315 substitution shall be considered to have been performed even if the replacement string is  
 13316 identical to the string that it replaces.

13317 The application shall ensure that the value of *flags* is zero or more of:

13318 *count* Substitute for the *count*th occurrence only of the RE found on each addressed line.

13319 **g** Globally substitute for all non-overlapping instances of the RE rather than just the first  
 13320 one. If both **g** and *count* are specified, the results are unspecified.

13321 **l** Write to standard output the final line in which a substitution was made. The line shall  
 13322 be written in the format specified for the **l** command.

13323 **n** Write to standard output the final line in which a substitution was made. The line shall  
 13324 be written in the format specified for the **n** command.

13325 **p** Write to standard output the final line in which a substitution was made. The line shall  
 13326 be written in the format specified for the **p** command.

## 13327 Copy Command

13328 *Synopsis:* (.,.)*taddress*

13329 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines  
 13330 shall be placed after address *address* (which can be 0); the current line number shall be set to the  
 13331 address of the last line added.

## 13332 Undo Command

13333 *Synopsis:* u

13334 The **u** command shall nullify the effect of the most recent command that modified anything in  
 13335 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes  
 13336 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no  
 13337 changes were made by the global command (such as with **g**/RE/**p**), the **u** command shall have  
 13338 no effect. The current line number shall be set to the value it had immediately before the  
 13339 command being undone started.

**Global Non-Matched Command**

*Synopsis:* (1,\$)v/RE/command list

This command shall be equivalent to the global command **g** except that the lines that are marked during the first step shall be those for which the line excluding the terminating <newline> does not match the RE.

**Interactive Global Not-Matched Command**

*Synopsis:* (1,\$)V/RE/

This command shall be equivalent to the interactive global command **G** except that the lines that are marked during the first step shall be those for which the line excluding the terminating <newline> does not match the RE.

**Write Command**

*Synopsis:* (1,\$)w [file]

The **w** command shall write the addressed lines into the file named by the pathname *file*. The command shall create the file, if it does not exist, or shall replace the contents of the existing file. The currently remembered pathname shall not be changed unless there is no remembered pathname. If no pathname is given, the currently remembered pathname, if any, shall be used (see the **e** and **f** commands); the current line number shall be unchanged. If the command is successful, the number of bytes written shall be written to standard output, unless the **-s** option was specified, in the following format:

"%d\n", <number of bytes written>

If *file* begins with '!', the rest of the line shall be taken to be a shell command line whose standard input shall be the addressed lines. Such a shell command line shall not be remembered as the current pathname. This usage of the write command with '!' shall not be considered as a "last **w** command that wrote the entire buffer", as described previously; thus, this alone shall not prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.

**Line Number Command**

*Synopsis:* (\$)=

The line number of the addressed line shall be written to standard output in the following format:

"%d\n", <line number>

The current line number shall be unchanged by this command.

**Shell Escape Command**

*Synopsis:* !command

The remainder of the line after the '!' shall be sent to the command interpreter to be interpreted as a shell command line. Within the text of that shell command line, the unescaped character '%' shall be replaced with the remembered pathname; if a '!' appears as the first character of the command, it shall be replaced with the text of the previous shell command executed via '!'. Thus, "!!" shall repeat the previous !command. If any replacements of '%' or '!' are performed, the modified line shall be written to the standard output before *command* is executed. The ! command shall write:

13381 "!\n"

13382 to standard output upon completion, unless the **-s** option is specified. The current line number  
13383 shall be unchanged.

#### 13384 **Null Command**

13385 *Synopsis:* ( . +1 )

13386 An address alone on a line shall cause the addressed line to be written. A <newline> alone shall  
13387 be equivalent to "+1p". The current line number shall be set to the address of the written line.

#### 13388 **EXIT STATUS**

13389 The following exit values shall be returned:

13390 0 Successful completion without any file or command errors.

13391 >0 An error occurred.

#### 13392 **CONSEQUENCES OF ERRORS**

13393 When an error in the input script is encountered, or when an error is detected that is a  
13394 consequence of the data (not) present in the file or due to an external condition such as a read or  
13395 write error:

13396 • If the standard input is a terminal device file, all input shall be flushed, and a new command  
13397 read.

13398 • If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.

#### 13399 **APPLICATION USAGE**

13400 Because of the extremely terse nature of the default error messages, the prudent script writer  
13401 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some  
13402 clue as to the cause is made available.

13403 In previous versions, an obsolescent **-** option was described. This is no longer specified.  
13404 Applications should use the **-s** option. Using **-** as a *file* operand now produces unspecified  
13405 results. This allows implementations to continue to support the former required behavior.

#### 13406 **EXAMPLES**

13407 None.

#### 13408 **RATIONALE**

13409 The initial description of this utility was adapted from the SVID. It contains some features not  
13410 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and  
13411 BSD *ed* utilities include, but need not be limited to:

13412 • The BSD **-** option does not suppress the '!' prompt after a **!** command.

13413 • BSD does not support the special meanings of the '%' and '!' characters within a **!**  
13414 command.

13415 • BSD does not support the *addresses* ';' and ', '.

13416 • BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this volume  
13417 of IEEE Std 1003.1-2001.

13418 • BSD does not support the '!' character part of the **e**, **r**, or **w** commands.

13419 • A failed **g** command in BSD sets the line number to the last line searched if there are no  
13420 matches.

- BSD does not default the *command list* to the **p** command.
- BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.
- On BSD, if there is no inserted text, the insert command changes the current line to the referenced line -1; that is, the line before the specified line.
- On BSD, the *join* command with only a single address changes the current line to that address.
- BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p** command.
- BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.
- The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in this volume of IEEE Std 1003.1-2001.

The **-s** option was added to allow the functionality of the now withdrawn **-** option in a manner compatible with the Utility Syntax Guidelines.

In early proposals there was a limit, {ED\_FILE\_MAX}, that described the historical limitations of some *ed* utilities in their handling of large files; some of these have had problems with files larger than 100 000 bytes. It was this limitation that prompted much of the desire to include a *split* command in this volume of IEEE Std 1003.1-2001. Since this limit was removed, this volume of IEEE Std 1003.1-2001 requires that implementations document the file size limits imposed by *ed* in the conformance document. The limit {ED\_LINE\_MAX} was also removed; therefore, the global limit {LINE\_MAX} is used for input and output lines.

The manner in which the **l** command writes non-printable characters was changed to avoid the historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous because most terminals simply replace overstruck characters, making the **l** format not useful for its intended purpose of unambiguously understanding the content of the line. The historical backslash escapes were also ambiguous. (The string "a\0011" could represent a line containing those six characters or a line containing the three characters 'a', a byte with a binary value of 1, and a 1.) In the format required here, a backslash appearing in the line is written as "\\ " so that the output is truly unambiguous. The method of marking the ends of lines was adopted from the *ex* editor and is required for any line ending in <space>s; the '\$' is placed on all lines so that a real '\$' at the end of a line cannot be misinterpreted.

Systems with bytes too large to fit into three octal digits must devise other means of displaying non-printable characters. Consideration was given to requiring that the number of octal digits be large enough to hold a byte, but this seemed to be too confusing for applications on the vast majority of systems where three digits are adequate. It would be theoretically possible for the application to use the *getconf* utility to find out the CHAR\_BIT value and deal with such an algorithm; however, there is really no portable way that an application can use the octal values of the bytes across various coded character sets, so the additional specification was not worthwhile.

The description of how a NUL is written was removed. The NUL character cannot be in text files, and this volume of IEEE Std 1003.1-2001 should not dictate behavior in the case of undefined, erroneous input.

Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands are not patterns.

Early proposals stated that the **-p** option worked only when standard input was associated with a terminal device. This has been changed to conform to historical implementations, thereby allowing applications to interpose themselves between a user and the *ed* utility.



13467 The form of the substitute command that uses the **n** suffix was limited in some historical  
 13468 documentation (where this was described incorrectly as “backreferencing”). This limit has been  
 13469 omitted because there is no reason why an editor processing lines of {LINE\_MAX} length should  
 13470 have this restriction. The command **s/x/X/2047** should be able to substitute the 2047th  
 13471 occurrence of ‘x’ on a line.

13472 The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made  
 13473 unspecified because BSD-based systems allow this, whereas System V does not.

13474 Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file  
 13475 have been deleted. Since this volume of IEEE Std 1003.1-2001 refers to the **q** command in this  
 13476 instance, such behavior is not allowed.

13477 Some historical implementations returned exit status zero even if command errors had occurred;  
 13478 this is not allowed by this volume of IEEE Std 1003.1-2001.

13479 Some historical implementations contained a bug that allowed a single period to be entered in  
 13480 input mode as <backslash> <period> <newline>. This is not allowed by *ed* because there is no  
 13481 description of escaping any of the characters in input mode; backslashes are entered into the  
 13482 buffer exactly as typed. The typical method of entering a single period has been to precede it  
 13483 with another character and then use the substitute command to delete that character.

13484 It is difficult under some modes of some versions of historical operating system terminal drivers  
 13485 to distinguish between an end-of-file condition and terminal disconnect. IEEE Std 1003.1-2001  
 13486 does not require implementations to distinguish between the two situations, which permits  
 13487 historical implementations of the *ed* utility on historical platforms to conform. Implementations  
 13488 are encouraged to distinguish between the two, if possible, and take appropriate action on  
 13489 terminal disconnect.

13490 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the  
 13491 start of the edit buffer. When the buffer was empty the command **.=** returned zero. IEEE Std  
 13492 1003.1-2001 requires conformance to historical practice.

13493 For consistency with the **a** and **r** commands and better user functionality, the **i** and **c** commands  
 13494 must also accept an address of 0, in which case **0i** is treated as **1i** and likewise for the **c**  
 13495 command.

13496 All of the following are valid addresses:

13497     +++               Three lines after the current line.

13498     /*pattern*/-       One line before the next occurrence of pattern.

13499     -2                Two lines before the current line.

13500     3 ---- 2          Line one (note the intermediate negative address).

13501     1 2 3            Line six.

13502 Any number of addresses can be provided to commands taking addresses; for example,  
 13503 "1,2,3,4,5p" prints lines 4 and 5, because two is the greatest valid number of addresses  
 13504 accepted by the **print** command. This, in combination with the semicolon delimiter, permits  
 13505 users to create commands based on ordered patterns in the file. For example, the command  
 13506 "3;/foo/;+2p" will display the first line after line 3 that contains the pattern *foo*, plus the next  
 13507 two lines. Note that the address "3;" must still be evaluated before being discarded, because  
 13508 the search origin for the "/foo/" command depends on this.

13509 Historically, *ed* disallowed address chains, as discussed above, consisting solely of comma or  
 13510 semicolon separators; for example, ",,," or ";;;" were considered an error. For consistency of  
 13511 address specification, this restriction is removed. The following table lists some of the address

13512 forms now possible:

| Address | Addr1 | Addr2 | Status     | Comment               |
|---------|-------|-------|------------|-----------------------|
| 7,      | 7     | 7     | Historical | Valid, but erroneous. |
| 7,5,    | 5     | 5     | Historical |                       |
| 7,5,9   | 5     | 9     | Historical |                       |
| 7,9     | 7     | 9     | Historical |                       |
| 7,+     | 7     | 8     | Historical |                       |
| ,       | 1     | \$    | Historical |                       |
| ,7      | 1     | 7     | Extension  |                       |
| ,,      | \$    | \$    | Extension  |                       |
| ,;      | \$    | \$    | Extension  |                       |
| 7;      | 7     | 7     | Historical |                       |
| 7;5;    | 5     | 5     | Historical |                       |
| 7;5;9   | 5     | 9     | Historical |                       |
| 7;5,9   | 5     | 9     | Historical |                       |
| 7;\$;4  | \$    | 4     | Historical |                       |
| 7;9     | 7     | 9     | Historical |                       |
| 7;+     | 7     | 8     | Historical |                       |
| ;       | .     | \$    | Historical |                       |
| ;7      | .     | 7     | Extension  |                       |
| ;;      | \$    | \$    | Extension  |                       |
| ;;,     | \$    | \$    | Extension  |                       |

13534 Historically, values could be added to addresses by including them after one or more <blank>s;  
 13535 for example, "3 – 5p" wrote the seventh line of the file, and "/f00/ 5" was the same as  
 13536 "5 /f00/". However, only absolute values could be added; for example, "5 /f00/" was an  
 13537 error. IEEE Std 1003.1-2001 requires conformance to historical practice.

13538 Historically, *ed* accepted the '^' character as an address, in which case it was identical to the  
 13539 hyphen character. IEEE Std 1003.1-2001 does not require or prohibit this behavior.

#### 13540 FUTURE DIRECTIONS

13541 None.

#### 13542 SEE ALSO

13543 Section 1.11 (on page 20), *ex*, *sed*, *sh*, *vi*

#### 13544 CHANGE HISTORY

13545 First released in Issue 2.

#### 13546 Issue 5

13547 In the OPTIONS section, the meaning of –s and – is clarified.

13548 A second FUTURE DIRECTION is added.

#### 13549 Issue 6

13550 The obsolescent single-minus form is removed.

13551 A second APPLICATION USAGE note is added.

13552 The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

13553 The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of  
 13554 the treatment of the SIGQUIT signal, changes to *ed* addressing, and changes to processing when  
 13555 end-of-file is detected and when terminal disconnect is detected.

|       |                                                                                                          |  |
|-------|----------------------------------------------------------------------------------------------------------|--|
| 13556 | The normative text is reworded to avoid use of the term “must” for application requirements.             |  |
| 13557 | IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/22 is applied, adding the text: “Any line             |  |
| 13558 | modified by the <i>command list</i> shall be unmarked.” to the <b>G</b> command. This change corresponds |  |
| 13559 | to a similar change made to the <b>g</b> command in the first version of IEEE Std 1003.1-2001.           |  |

13560 **NAME**

13561       env — set the environment for command invocation

13562 **SYNOPSIS**13563       env [-i] [*name=value*]... [*utility* [*argument...*]]13564 **DESCRIPTION**13565       The *env* utility shall obtain the current environment, modify it according to its arguments, then  
13566       invoke the utility named by the *utility* operand with the modified environment.13567       Optional arguments shall be passed to *utility*.13568       If no *utility* operand is specified, the resulting environment shall be written to the standard  
13569       output, with one *name=value* pair per line.13570 **OPTIONS**13571       The *env* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
13572       12.2, Utility Syntax Guidelines.

13573       The following options shall be supported:

13574       -**i**           Invoke *utility* with exactly the environment specified by the arguments; the  
13575       inherited environment shall be ignored completely.13576 **OPERANDS**

13577       The following operands shall be supported:

13578       *name=value*   Arguments of the form *name=value* shall modify the execution environment, and  
13579       shall be placed into the inherited environment before the *utility* is invoked.13580       *utility*       The name of the utility to be invoked. If the *utility* operand names any of the  
13581       special built-in utilities in Section 2.14 (on page 64), the results are undefined.13582       *argument*    A string to pass as an argument for the invoked utility.13583 **STDIN**

13584       Not used.

13585 **INPUT FILES**

13586       None.

13587 **ENVIRONMENT VARIABLES**13588       The following environment variables shall affect the execution of *env*:13589       **LANG**        Provide a default value for the internationalization variables that are unset or null.  
13590       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
13591       Internationalization Variables for the precedence of internationalization variables  
13592       used to determine the values of locale categories.)13593       **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
13594       internationalization variables.13595       **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
13596       characters (for example, single-byte as opposed to multi-byte characters in  
13597       arguments).13598       **LC\_MESSAGES**13599       Determine the locale that should be used to affect the format and contents of  
13600       diagnostic messages written to standard error.13601 **XS1**       **NLS\_PATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

13602            *PATH*            Determine the location of the *utility*, as described in the Base Definitions volume of  
 13603                                IEEE Std 1003.1-2001, Chapter 8, Environment Variables. If *PATH* is specified as a  
 13604                                *name=value* operand to *env*, the *value* given shall be used in the search for *utility*.

#### 13605 **ASYNCHRONOUS EVENTS**

13606            Default.

#### 13607 **STDOUT**

13608            If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be  
 13609                                written in the form:

13610            "%s=%s\n", <*name*>, <*value*>

13611            If the *utility* operand is specified, the *env* utility shall not write to standard output.

#### 13612 **STDERR**

13613            The standard error shall be used only for diagnostic messages.

#### 13614 **OUTPUT FILES**

13615            None.

#### 13616 **EXTENDED DESCRIPTION**

13617            None.

#### 13618 **EXIT STATUS**

13619            If *utility* is invoked, the exit status of *env* shall be the exit status of *utility*; otherwise, the *env*  
 13620                                utility shall exit with one of the following values:

13621            0        The *env* utility completed successfully.

13622            1–125    An error occurred in the *env* utility.

13623            126     The utility specified by *utility* was found but could not be invoked.

13624            127     The utility specified by *utility* could not be found.

#### 13625 **CONSEQUENCES OF ERRORS**

13626            Default.

#### 13627 **APPLICATION USAGE**

13628            The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
 13629                                an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
 13630                                utility exited with an error indication”. The value 127 was chosen because it is not commonly  
 13631                                used for other meanings; most utilities use small values for “normal error conditions” and the  
 13632                                values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
 13633                                chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
 13634                                scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
 13635                                between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
 13636                                *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
 13637                                any other reason.

13638            Historical implementations of the *env* utility use the *execvp()* or *execlp()* functions defined in the  
 13639                                System Interfaces volume of IEEE Std 1003.1-2001 to invoke the specified utility; this provides  
 13640                                better performance and keeps users from having to escape characters with special meaning to  
 13641                                the shell. Therefore, shell functions, special built-ins, and built-ins that are only provided by the  
 13642                                shell are not found.

**13643 EXAMPLES**

13644       The following command:

13645       env -i PATH=/mybin mygrep xyz myfile

13646       invokes the command *mygrep* with a new *PATH* value as the only entry in its environment. In  
13647       this case, *PATH* is used to locate *mygrep*, which then must reside in **/mybin**.

**13648 RATIONALE**

13649       As with all other utilities that invoke other utilities, this volume of IEEE Std 1003.1-2001 only  
13650       specifies what *env* does with standard input, standard output, standard error, input files, and  
13651       output files. If a utility is executed, it is not constrained by the specification of input and output  
13652       by *env*.

13653       The **-i** option was added to allow the functionality of the withdrawn **-** option in a manner  
13654       compatible with the Utility Syntax Guidelines.

13655       Some have suggested that *env* is redundant since the same effect is achieved by:

13656       name=value ... utility [ argument ... ]

13657       The example is equivalent to *env* when an environment variable is being added to the  
13658       environment of the command, but not when the environment is being set to the given value.

13659       The *env* utility also writes out the current environment if invoked without arguments. There is  
13660       sufficient functionality beyond what the example provides to justify inclusion of *env*.

**13661 FUTURE DIRECTIONS**

13662       None.

**13663 SEE ALSO**

13664       Section 2.5 (on page 33), Section 2.14 (on page 64)

**13665 CHANGE HISTORY**

13666       First released in Issue 2.

13667 **NAME**

13668        ex — text editor

13669 **SYNOPSIS**13670 UP        ex [-rR] [-s | -v] [-c *command*] [-t *tagstring*] [-w *size*] [*file* ...]

13671

13672 **DESCRIPTION**

13673        The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and  
 13674        visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open**  
 13675        and **visual** commands and in *vi*.

13676        This section uses the term *edit buffer* to describe the current working text. No specific  
 13677        implementation is implied by this term. All editing changes are performed on the edit buffer,  
 13678        and no changes to it shall affect any file until an editor command writes the file.

13679        Certain terminals do not have all the capabilities necessary to support the complete *ex* definition,  
 13680        such as the full-screen editing commands (*visual mode* or *open mode*). When these commands  
 13681        cannot be supported on such terminals, this condition shall not produce an error message such  
 13682        as “not an editor command” or report a syntax error. The implementation may either accept the  
 13683        commands and produce results on the screen that are the result of an unsuccessful attempt to  
 13684        meet the requirements of this volume of IEEE Std 1003.1-2001 or report an error describing the  
 13685        terminal-related deficiency.

13686 **OPTIONS**

13687        The *ex* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 13688        Utility Syntax Guidelines.

13689        The following options shall be supported:

13690        **-c *command*** Specify an initial command to be executed in the first edit buffer loaded from an  
 13691        existing file (see the EXTENDED DESCRIPTION section). Implementations may  
 13692        support more than a single **-c** option. In such implementations, the specified  
 13693        commands shall be executed in the order specified on the command line.

13694        **-r** Recover the named files (see the EXTENDED DESCRIPTION section). Recovery  
 13695        information for a file shall be saved during an editor or system crash (for example,  
 13696        when the editor is terminated by a signal which the editor can catch), or after the  
 13697        use of an *ex* **preserve** command.

13698        A *crash* in this context is an unexpected failure of the system or utility that requires  
 13699        restarting the failed system or utility. A system crash implies that any utilities  
 13700        running at the time also crash. In the case of an editor or system crash, the number  
 13701        of changes to the edit buffer (since the most recent **preserve** command) that will be  
 13702        recovered is unspecified.

13703        If no *file* operands are given and the **-t** option is not specified, all other options, the  
 13704        *EXINIT* variable, and any **.exrc** files shall be ignored; a list of all recoverable files  
 13705        available to the invoking user shall be written, and the editor shall exit normally  
 13706        without further action.

13707        **-R** Set **readonly** edit option.

13708        **-s** Prepare *ex* for batch use by taking the following actions:

- 13709           • Suppress writing prompts and informational (but not diagnostic) messages.
- 13710           • Ignore the value of *TERM* and any implementation default terminal type and
- 13711           assume the terminal is a type incapable of supporting open or visual modes;

|       |                              |                                                                                                                |
|-------|------------------------------|----------------------------------------------------------------------------------------------------------------|
| 13712 |                              | see the <b>visual</b> command and the description of <i>vi</i> .                                               |
| 13713 |                              | • Suppress the use of the <i>EXINIT</i> environment variable and the reading of any                            |
| 13714 |                              | <b>.exrc</b> file; see the EXTENDED DESCRIPTION section.                                                       |
| 13715 |                              | • Suppress autoindentation, ignoring the value of the <b>autoindent</b> edit option.                           |
| 13716 | <b>-t tagstring</b>          | Edit the file containing the specified <i>tagstring</i> ; see <i>ctags</i> . The tags feature                  |
| 13717 |                              | represented by <b>-t tagstring</b> and the <b>tag</b> command is optional. It shall be provided                |
| 13718 |                              | on any system that also provides a conforming implementation of <i>ctags</i> ; otherwise,                      |
| 13719 |                              | the use of <b>-t</b> produces undefined results. On any system, it shall be an error to                        |
| 13720 |                              | specify more than a single <b>-t</b> option.                                                                   |
| 13721 | <b>-v</b>                    | Begin in visual mode (see <i>vi</i> ).                                                                         |
| 13722 | <b>-w size</b>               | Set the value of the <i>window</i> editor option to <i>size</i> .                                              |
| 13723 | <b>OPERANDS</b>              |                                                                                                                |
| 13724 |                              | The following operand shall be supported:                                                                      |
| 13725 | <i>file</i>                  | A pathname of a file to be edited.                                                                             |
| 13726 | <b>STDIN</b>                 |                                                                                                                |
| 13727 |                              | The standard input consists of a series of commands and input text, as described in the                        |
| 13728 |                              | EXTENDED DESCRIPTION section. The implementation may limit each line of standard input                         |
| 13729 |                              | to a length of { <i>LINE_MAX</i> }.                                                                            |
| 13730 |                              | If the standard input is not a terminal device, it shall be as if the <b>-s</b> option had been specified.     |
| 13731 |                              | If a read from the standard input returns an error, or if the editor detects an end-of-file condition          |
| 13732 |                              | from the standard input, it shall be equivalent to a SIGHUP asynchronous event.                                |
| 13733 | <b>INPUT FILES</b>           |                                                                                                                |
| 13734 |                              | Input files shall be text files or files that would be text files except for an incomplete last line that      |
| 13735 |                              | is not longer than { <i>LINE_MAX</i> }-1 bytes in length and contains no NUL characters. By default,           |
| 13736 |                              | any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other              |
| 13737 |                              | forms of files may optionally be allowed by <i>ex</i> implementations.                                         |
| 13738 |                              | The <b>.exrc</b> files and source files shall be text files consisting of <i>ex</i> commands; see the EXTENDED |
| 13739 |                              | DESCRIPTION section.                                                                                           |
| 13740 |                              | By default, the editor shall read lines from the files to be edited without interpreting any of those          |
| 13741 |                              | lines as any form of editor command.                                                                           |
| 13742 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                |
| 13743 |                              | The following environment variables shall affect the execution of <i>ex</i> :                                  |
| 13744 | <b>COLUMNS</b>               | Override the system-selected horizontal screen size. See the Base Definitions                                  |
| 13745 |                              | volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables for valid values                              |
| 13746 |                              | and results when it is unset or null.                                                                          |
| 13747 | <b>EXINIT</b>                | Determine a list of <i>ex</i> commands that are executed on editor start-up. See the                           |
| 13748 |                              | EXTENDED DESCRIPTION section for more details of the initialization phase.                                     |
| 13749 | <b>HOME</b>                  | Determine a pathname of a directory that shall be searched for an editor start-up                              |
| 13750 |                              | file named <b>.exrc</b> ; see the EXTENDED DESCRIPTION section.                                                |
| 13751 | <b>LANG</b>                  | Provide a default value for the internationalization variables that are unset or null.                         |
| 13752 |                              | (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,                                         |
| 13753 |                              | Internationalization Variables for the precedence of internationalization variables                            |
| 13754 |                              | used to determine the values of locale categories.)                                                            |



|       |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------|--------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 13755 | <i>LC_ALL</i>                                                                                                | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                          |
| 13756 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13757 | <i>LC_COLLATE</i>                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13758 |                                                                                                              | Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.                                                                                                                                                                                                                                                          |
| 13759 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13760 | <i>LC_CTYPE</i>                                                                                              | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as uppercase or lowercase letters, the case conversion of letters, and the detection of word boundaries. |
| 13761 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13762 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13763 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13764 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13765 | <i>LC_MESSAGES</i>                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13766 |                                                                                                              | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                      |
| 13767 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13768 | <i>LINES</i>                                                                                                 | Override the system-selected vertical screen size, used as the number of lines in a screenful and the vertical screen size in visual mode. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables for valid values and results when it is unset or null.                                                                                                       |
| 13769 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13770 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13771 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13772 | <i>NSLSPATH</i>                                                                                              | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                             |
| 13773 | <i>PATH</i>                                                                                                  | Determine the search path for the shell command specified in the <i>ex</i> editor commands <b>!</b> , <b>shell</b> , <b>read</b> , and <b>write</b> , and the open and visual mode command <b>!</b> ; see the description of command search and execution in Section 2.9.1.1 (on page 48).                                                                                                        |
| 13774 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13775 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13776 | <i>SHELL</i>                                                                                                 | Determine the preferred command line interpreter for use as the default value of the <b>shell</b> edit option.                                                                                                                                                                                                                                                                                    |
| 13777 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13778 | <i>TERM</i>                                                                                                  | Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.                                                                                                                                                                                                                                                                   |
| 13779 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13780 | <b>ASYNCHRONOUS EVENTS</b>                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13781 | The following term is used in this and following sections to specify command and asynchronous event actions: |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13782 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13783 | <i>complete write</i>                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13784 |                                                                                                              | A complete write is a write of the entire contents of the edit buffer to a file of a type other than a terminal device, or the saving of the edit buffer caused by the user executing the <i>ex</i> <b>preserve</b> command. Writing the contents of the edit buffer to a temporary file that will be removed when the editor exits shall not be considered a complete write.                     |
| 13785 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13786 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13787 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13788 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13789 | The following actions shall be taken upon receipt of signals:                                                |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13790 | <i>SIGINT</i>                                                                                                | If the standard input is not a terminal device, <i>ex</i> shall not write the file or return to command or text input mode, and shall exit with a non-zero exit status.                                                                                                                                                                                                                           |
| 13791 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13792 |                                                                                                              | Otherwise, if executing an open or visual text input mode command, <i>ex</i> in receipt of <i>SIGINT</i> shall behave identically to its receipt of the <ESC> character.                                                                                                                                                                                                                          |
| 13793 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13794 |                                                                                                              | Otherwise:                                                                                                                                                                                                                                                                                                                                                                                        |
| 13795 |                                                                                                              | 1. If executing an <i>ex</i> text input mode command, all input lines that have been completely entered shall be resolved into the edit buffer, and any partially entered line shall be discarded.                                                                                                                                                                                                |
| 13796 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13797 |                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                   |

- 13798                   2. If there is a currently executing command, it shall be aborted and a message  
13799                   displayed. Unless otherwise specified by the *ex* or *vi* command descriptions,  
13800                   it is unspecified whether any lines modified by the executing command  
13801                   appear modified, or as they were before being modified by the executing  
13802                   command, in the buffer.
- 13803                   If the currently executing command was a motion command, its associated  
13804                   command shall be discarded.
- 13805                   3. If in open or visual command mode, the terminal shall be alerted.
- 13806                   4. The editor shall then return to command mode.
- 13807           SIGCONT   The screen shall be refreshed if in open or visual mode.
- 13808           SIGHUP    If the edit buffer has been modified since the last complete write, *ex* shall attempt  
13809           to save the edit buffer so that it can be recovered later using the **-r** option or the *ex*  
13810           **recover** command. The editor shall not write the file or return to command or text  
13811           input mode, and shall terminate with a non-zero exit status.
- 13812           SIGTERM   Refer to SIGHUP.
- 13813           The action taken for all other signals is unspecified.
- 13814   **STDOUT**
- 13815           The standard output shall be used only for writing prompts to the user, for informational  
13816           messages, and for writing lines from the file.
- 13817   **STDERR**
- 13818           The standard error shall be used only for diagnostic messages.
- 13819   **OUTPUT FILES**
- 13820           The output from *ex* shall be text files.
- 13821   **EXTENDED DESCRIPTION**
- 13822           Only the *ex* mode of the editor is described in this section. See *vi* for additional editing  
13823           capabilities available in *ex*.
- 13824           When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such  
13825           as inverse video), the message shall be written in standout mode. If the terminal does not  
13826           support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the  
13827           error message.
- 13828           By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the  
13829           **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;  
13830           it can be exited (and command mode re-entered) by typing a period ( **.** ) alone at the beginning  
13831           of a line.
- 13832   **Initialization in ex and vi**
- 13833           The following symbols are used in this and following sections to specify locations in the edit  
13834           buffer:
- 13835           *alternate and current pathnames*
- 13836           Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex*  
13837           commands that take filenames as arguments shall set them as follows:
- 13838           1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag**  
13839           command replaces the contents of the edit buffer.

- 13840                   a. If the command replaces the contents of the edit buffer, the current pathname  
13841                   shall be set to the *file* argument or the file indicated by the tag, and the alternate  
13842                   pathname shall be set to the previous value of the current pathname.
- 13843                   b. Otherwise, the alternate pathname shall be set to the *file* argument.
- 13844           2. If a *file* argument is specified to the **ex next** command:
- 13845                   a. If the command replaces the contents of the edit buffer, the current pathname  
13846                   shall be set to the first *file* argument, and the alternate pathname shall be set to  
13847                   the previous value of the current pathname.
- 13848           3. If a *file* argument is specified to the **ex file** command, the current pathname shall be set  
13849           to the *file* argument, and the alternate pathname shall be set to the previous value of  
13850           the current pathname.
- 13851           4. If a *file* argument is specified to the **ex read** and **write** commands (that is, when  
13852           reading or writing a file, and not to the program named by the **shell** edit option), or a  
13853           *file* argument is specified to the **ex xit** command:
- 13854                   a. If the current pathname has no value, the current pathname shall be set to the *file*  
13855                   argument.
- 13856                   b. Otherwise, the alternate pathname shall be set to the *file* argument.

13857           If the alternate pathname is set to the previous value of the current pathname when the  
13858           current pathname had no previous value, then the alternate pathname shall have no value  
13859           as a result.

#### 13860 *current line*

13861           The line of the edit buffer referenced by the cursor. Each command description specifies the  
13862           current line after the command has been executed, as the *current line value*. When the edit  
13863           buffer contains no lines, the current line shall be zero; see **Addressing in ex** (on page 361).

#### 13864 *current column*

13865           The current display line column occupied by the cursor. (The columns shall be numbered  
13866           beginning at 1.) Each command description specifies the current column after the command  
13867           has been executed, as the *current column value*. This column is an *ideal* column that is  
13868           remembered over the lifetime of the editor. The actual display line column upon which the  
13869           cursor rests may be different from the current column; see the cursor positioning discussion  
13870           in **Command Descriptions in vi** (on page 988).

#### 13871 *set to non-<blank>*

13872           A description for a current column value, meaning that the current column shall be set to  
13873           the last display line column on which is displayed any part of the first non-<blank> of the  
13874           line. If the line has no non-<blank> non-<newline>s, the current column shall be set to the  
13875           last display line column on which is displayed any part of the last non-<newline> in the  
13876           line. If the line is empty, the current column shall be set to column position 1.

13877           The length of lines in the edit buffer may be limited to {LINE\_MAX} bytes. In open and visual  
13878           mode, the length of lines in the edit buffer may be limited to the number of characters that will  
13879           fit in the display. If either limit is exceeded during editing, an error message shall be written. If  
13880           either limit is exceeded by a line read in from a file, an error message shall be written and the  
13881           edit session may be terminated.

13882           If the editor stops running due to any reason other than a user command, and the edit buffer has  
13883           been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous  
13884           event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

During initialization (before the first file is copied into the edit buffer or any user commands from the terminal are processed) the following shall occur:

1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands contained in that variable.
2. If the *EXINIT* variable is not set, and all of the following are true:
  - a. The *HOME* environment variable is not null and not empty.
  - b. The file **.exrc** in the directory referred to by the *HOME* environment variable:
    1. Exists
    2. Is owned by the same user ID as the real user ID of the process or the process has appropriate privileges
    3. Is not writable by anyone other than the owner

the editor shall execute the *ex* commands contained in that file.

3. If and only if all of the following are true:
  - a. The current directory is not referred to by the *HOME* environment variable.
  - b. A command in the *EXINIT* environment variable or a command in the **.exrc** file in the directory referred to by the *HOME* environment variable sets the editor option **exrc**.
  - c. The **.exrc** file in the current directory:
    1. Exists
    2. Is owned by the same user ID as the real user ID of the process, or by one of a set of implementation-defined user IDs
    3. Is not writable by anyone other than the owner

the editor shall attempt to execute the *ex* commands contained in that file.

Lines in any **.exrc** file that are blank lines shall be ignored. If any **.exrc** file exists, but is not read for ownership or permission reasons, it shall be an error.

After the *EXINIT* variable and any **.exrc** files are processed, the first file specified by the user shall be edited, as follows:

1. If the user specified the **-t** option, the effect shall be as if the *ex* **tag** command was entered with the specified argument, with the exception that if tag processing does not result in a file to edit, the effect shall be as described in step 3. below.
2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if the *ex* **edit** command was entered with the first of those arguments as its *file* argument.
3. Otherwise, the effect shall be as if the *ex* **edit** command was entered with a nonexistent filename as its *file* argument. It is unspecified whether this action shall set the current pathname. In an implementation where this action does not set the current pathname, any editor command using the current pathname shall fail until an editor command sets the current pathname.

If the **-r** option was specified, the first time a file in the initial argument list or a file specified by the **-t** option is edited, if recovery information has previously been saved about it, that information shall be recovered and the editor shall behave as if the contents of the edit buffer have already been modified. If there are multiple instances of the file to be recovered, the one most recently saved shall be recovered, and an informational message that there are previous

versions of the file that can be recovered shall be written. If no recovery information about a file is available, an informational message to this effect shall be written, and the edit shall proceed as usual.

If the **-c** option was specified, the first time a file that already exists (including a file that might not exist but for which recovery information is available, when the **-r** option is specified) replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of the edit buffer, the current column shall be set to non-<blank>, and the *ex* commands specified with the **-c** option shall be executed. In this case, the current line and current column shall not be set as described for the command associated with the replacement or initialization of the edit buffer contents. However, if the **-t** option or a **tag** command is associated with this action, the **-c** option commands shall be executed and then the movement to the tag shall be performed.

The current argument list shall initially be set to the filenames specified by the user on the command line. If no filenames are specified by the user, the current argument list shall be empty. If the **-t** option was specified, it is unspecified whether any filename resulting from tag processing shall be prepended to the current argument list. In the case where the filename is added as a prefix to the current argument list, the current argument list reference shall be set to that filename. In the case where the filename is not added as a prefix to the current argument list, the current argument list reference shall logically be located before the first of the filenames specified on the command line (for example, a subsequent *ex* **next** command shall edit the first filename from the command line). If the **-t** option was not specified, the current argument list reference shall be to the first of the filenames on the command line.

### Addressing in *ex*

Addressing in *ex* relates to the current line and the current column; the address of a line is its 1-based line number, the address of a column is its 1-based count from the beginning of the line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. In each command description, the effect of the command on the current line number and the current column is described.

Addresses are constructed as follows:

1. The character **'.'** (period) shall address the current line.
2. The character **'\$'** shall address the last line of the edit buffer.
3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
4. The address **"'x"** refers to the line marked with the mark name character **'x'**, which shall be a lowercase letter from the portable character set or one of the characters **'\''** or **'\'''**. It shall be an error if the line that was marked is not currently present in the edit buffer or the mark has not been set. Lines can be marked with the *ex* **mark** or **k** commands, or the *vi* **m** command.
5. A regular expression enclosed by slashes (**'/'**) shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the regular expression. As stated in **Regular Expressions in *ex*** (on page 391), an address consisting of a null regular expression delimited by slashes **"/"** shall address the next line for which the line excluding the terminating <newline> matches the last regular expression encountered. In addition, the second slash can be omitted at the end of a command line. If the **wrapsan** edit option is set, the search shall wrap around to the beginning of the edit buffer and continue up to and including the current line, so that the entire edit buffer is searched. Within the regular expression, the sequence **"\"** shall represent a literal slash instead of the regular expression delimiter.

6. A regular expression enclosed in question marks ('?') shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the regular expression. An address consisting of a null regular expression delimited by question marks "?? " shall address the previous line for which the line excluding the terminating <newline> matches the last regular expression encountered. In addition, the second question mark can be omitted at the end of a command line. If the **wrapsan** edit option is set, the search shall wrap around from the beginning of the edit buffer to the end of the edit buffer and continue up to and including the current line, so that the entire edit buffer is searched. Within the regular expression, the sequence "\?" shall represent a literal question mark instead of the RE delimiter.

7. A plus sign ('+') or a minus sign ('-') followed by a decimal number shall address the current line plus or minus the number. A '+' or '-' not followed by a decimal number shall address the current line plus or minus 1.

Addresses can be followed by zero or more address offsets, optionally <blank>-separated. Address offsets are constructed as follows:

1. A '+' or '-' immediately followed by a decimal number shall add (subtract) the indicated number of lines to (from) the address. A '+' or '-' not followed by a decimal number shall add (subtract) 1 to (from) the address.
2. A decimal number shall add the indicated number of lines to the address.

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer.

Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in **Command Descriptions in ex** (on page 368). If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first shall be evaluated and then discarded until the maximum number of valid addresses remain.

Addresses shall be separated from each other by a comma (',') or a semicolon(';'). If no address is specified before or after a comma or semicolon separator, it shall be as if the address of the current line was specified before or after the separator. In the case of a semicolon separator, the current line ('.') shall be set to the first address, and only then will the next address be calculated. This feature can be used to determine the starting line for forwards and backwards searches (see rules 5. and 6.).

A percent sign('%') shall be equivalent to entering the two addresses "1, \$".

Any delimiting <blank>s between addresses, address separators, or address offsets shall be discarded.

### Command Line Parsing in ex

The following symbol is used in this and following sections to describe parsing behavior:

*escape* If a character is referred to as “backslash-escaped” or “<control>-V-escaped,” it shall mean that the character acquired or lost a special meaning by virtue of being preceded, respectively, by a backslash or <control>-V character. Unless otherwise specified, the escaping character shall be discarded at that time and shall not be further considered for any purpose.

14017 Command-line parsing shall be done in the following steps. For each step, characters already  
 14018 evaluated shall be ignored; that is, the phrase “leading character” refers to the next character  
 14019 that has not yet been evaluated.

- 14020 1. Leading colon characters shall be skipped.
- 14021 2. Leading <blank>s shall be skipped.
- 14022 3. If the leading character is a double-quote character, the characters up to and including the  
 14023 next non-backslash-escaped <newline> shall be discarded, and any subsequent characters  
 14024 shall be parsed as a separate command.
- 14025 4. Leading characters that can be interpreted as addresses shall be evaluated; see **Addressing**  
 14026 **in ex** (on page 361).
- 14027 5. Leading <blank>s shall be skipped.
- 14028 6. If the next character is a vertical-line character or a <newline>:
  - 14029 a. If the next character is a <newline>:
    - 14030 1. If *ex* is in open or visual mode, the current line shall be set to the last address  
 14031 specified, if any.
    - 14032 2. Otherwise, if the last command was terminated by a vertical-line character, no  
 14033 action shall be taken; for example, the command "||<newline>" shall  
 14034 execute two implied commands, not three.
    - 14035 3. Otherwise, step 6.b. shall apply.
  - 14036 b. Otherwise, the implied command shall be the **print** command. The last **#**, **p**, and **l**  
 14037 flags specified to any *ex* command shall be remembered and shall apply to this  
 14038 implied command. Executing the *ex* **number**, **print**, or **list** command shall set the  
 14039 remembered flags to **#**, nothing, and **l**, respectively, plus any other flags specified for  
 14040 that execution of the **number**, **print**, or **list** command.  
  
 14041 If *ex* is not currently performing a **global** or **v** command, and no address or count is  
 14042 specified, the current line shall be incremented by 1 before the command is executed.  
 14043 If incrementing the current line would result in an address past the last line in the  
 14044 edit buffer, the command shall fail, and the increment shall not happen.
  - 14045 c. The <newline> or vertical-line character shall be discarded and any subsequent  
 14046 characters shall be parsed as a separate command.
- 14047 7. The command name shall be comprised of the next character (if the character is not  
 14048 alphabetic), or the next character and any subsequent alphabetic characters (if the  
 14049 character is alphabetic), with the following exceptions:
  - 14050 a. Commands that consist of any prefix of the characters in the command name **delete**,  
 14051 followed immediately by any of the characters 'l', 'p', '+', '-', or '#' shall be  
 14052 interpreted as a **delete** command, followed by a <blank>, followed by the characters  
 14053 that were not part of the prefix of the **delete** command. The maximum number of  
 14054 characters shall be matched to the command name **delete**; for example, "del" shall  
 14055 not be treated as "de" followed by the flag **l**.
  - 14056 b. Commands that consist of the character 'k', followed by a character that can be  
 14057 used as the name of a mark, shall be equivalent to the mark command followed by a  
 14058 <blank>, followed by the character that followed the 'k'.
  - 14059 c. Commands that consist of the character 's', followed by characters that could be  
 14060 interpreted as valid options to the **s** command, shall be the equivalent of the **s**

14061 command, without any pattern or replacement values, followed by a <blank>,  
14062 followed by the characters after the 's'.

- 14063 8. The command name shall be matched against the possible command names, and a  
14064 command name that contains a prefix matching the characters specified by the user shall  
14065 be the executed command. In the case of commands where the characters specified by the  
14066 user could be ambiguous, the executed command shall be as follows:

|       |           |               |           |              |           |              |
|-------|-----------|---------------|-----------|--------------|-----------|--------------|
| 14067 | <b>a</b>  | <b>append</b> | <b>n</b>  | <b>next</b>  | <b>t</b>  | <b>t</b>     |
| 14068 | <b>c</b>  | <b>change</b> | <b>p</b>  | <b>print</b> | <b>u</b>  | <b>undo</b>  |
| 14069 | <b>ch</b> | <b>change</b> | <b>pr</b> | <b>print</b> | <b>un</b> | <b>undo</b>  |
| 14070 | <b>e</b>  | <b>edit</b>   | <b>r</b>  | <b>read</b>  | <b>v</b>  | <b>v</b>     |
| 14071 | <b>m</b>  | <b>move</b>   | <b>re</b> | <b>read</b>  | <b>w</b>  | <b>write</b> |
| 14072 | <b>ma</b> | <b>mark</b>   | <b>s</b>  | <b>s</b>     |           |              |

14073 Implementation extensions with names causing similar ambiguities shall not be checked  
14074 for a match until all possible matches for commands specified by IEEE Std 1003.1-2001  
14075 have been checked.

- 14076 9. If the command is a **!** command, or if the command is a **read** command followed by zero  
14077 or more <blank>s and a **!**, or if the command is a **write** command followed by one or more  
14078 <blank>s and a **!**, the rest of the command shall include all characters up to a non-  
14079 backslash-escaped <newline>. The <newline> shall be discarded and any subsequent  
14080 characters shall be parsed as a separate *ex* command.

- 14081 10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while in  
14082 open or visual mode, the next part of the command shall be parsed as follows:

- 14083 a. Any '!' character immediately following the command shall be skipped and be part  
14084 of the command.
- 14085 b. Any leading <blank>s shall be skipped and be part of the command.
- 14086 c. If the next character is a '+', characters up to the first non-backslash-escaped  
14087 <newline> or non-backslash-escaped <blank> shall be skipped and be part of the  
14088 command.
- 14089 d. The rest of the command shall be determined by the steps specified in paragraph 12.

- 14090 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the  
14091 command shall be parsed as follows:

- 14092 a. Any leading <blank>s shall be skipped and be part of the command.
- 14093 b. If the next character is not an alphanumeric, double-quote, <newline>, backslash, or  
14094 vertical-line character:
- 14095 1. The next character shall be used as a command delimiter.
- 14096 2. If the command is a **global**, **open**, or **v** command, characters up to the first  
14097 non-backslash-escaped <newline>, or first non-backslash-escaped delimiter  
14098 character, shall be skipped and be part of the command.
- 14099 3. If the command is an **s** command, characters up to the first non-backslash-  
14100 escaped <newline>, or second non-backslash-escaped delimiter character, shall  
14101 be skipped and be part of the command.
- 14102 c. If the command is a **global** or **v** command, characters up to the first non-backslash-  
14103 escaped <newline> shall be skipped and be part of the command.



14104 d. Otherwise, the rest of the command shall be determined by the steps specified in  
14105 paragraph 12.

14106 12. Otherwise:

14107 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,  
14108 characters up to the first non-`<control>-V`-escaped `<newline>`, vertical-line, or  
14109 double-quote character shall be skipped and be part of the command.

14110 b. Otherwise, characters up to the first non-backslash-escaped `<newline>`, vertical-line,  
14111 or double-quote character shall be skipped and be part of the command.

14112 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.  
14113 ended at a vertical-line character, any subsequent characters, up to the next non-  
14114 backslash-escaped `<newline>` shall be used as input text to the command.

14115 d. If the command was ended by a double-quote character, all subsequent characters,  
14116 up to the next non-backslash-escaped `<newline>`, shall be discarded.

14117 e. The terminating `<newline>` or vertical-line character shall be discarded and any  
14118 subsequent characters shall be parsed as a separate *ex* command.

14119 Command arguments shall be parsed as described by the Synopsis and Description of each  
14120 individual *ex* command. This parsing shall not be `<blank>`-sensitive, except for the `!` argument,  
14121 which must follow the command name without intervening `<blank>`s, and where it would  
14122 otherwise be ambiguous. For example, *count* and *flag* arguments need not be `<blank>`-separated  
14123 because `"d22p"` is not ambiguous, but *file* arguments to the *ex* **next** command must be  
14124 separated by one or more `<blank>`s. Any `<blank>` in command arguments for the **abbreviate**,  
14125 **unabbreviate**, **map**, and **unmap** commands can be `<control>-V`-escaped, in which case the  
14126 `<blank>` shall not be used as an argument delimiter. Any `<blank>` in the command argument for  
14127 any other command can be backslash-escaped, in which case that `<blank>` shall not be used as  
14128 an argument delimiter.

14129 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,  
14130 any character can be `<control>-V`-escaped. All such escaped characters shall be treated literally  
14131 and shall have no special meaning. Within command arguments for all other *ex* commands that  
14132 are not regular expressions or replacement strings, any character that would otherwise have a  
14133 special meaning can be backslash-escaped. Escaped characters shall be treated literally, without  
14134 special meaning as shell expansion characters or `'!'`, `'%'`, and `'#'` expansion characters. See  
14135 **Regular Expressions in ex** (on page 391) and **Replacement Strings in ex** (on page 391) for  
14136 descriptions of command arguments that are regular expressions or replacement strings.

14137 Non-backslash-escaped `'%'` characters appearing in *file* arguments to any *ex* command shall be  
14138 replaced by the current pathname; unescaped `'#'` characters shall be replaced by the alternate  
14139 pathname. It shall be an error if `'%'` or `'#'` characters appear unescaped in an argument and  
14140 their corresponding values are not set.

14141 Non-backslash-escaped `'!'` characters in the arguments to either the *ex* `!` command or the open  
14142 and visual mode `!` command, or in the arguments to the *ex* **read** command, where the first non-  
14143 `<blank>` after the command name is a `'!'` character, or in the arguments to the *ex* **write**  
14144 command where the command name is followed by one or more `<blank>`s and the first non-  
14145 `<blank>` after the command name is a `'!'` character, shall be replaced with the arguments to the  
14146 last of those three commands as they appeared after all unescaped `'%'`, `'#'`, and `'!'` characters  
14147 were replaced. It shall be an error if `'!'` characters appear unescaped in one of these commands  
14148 and there has been no previous execution of one of these commands.

14149 If an error occurs during the parsing or execution of an *ex* command:

- An informational message to this effect shall be written. Execution of the *ex* command shall stop, and the cursor (for example, the current line and column) shall not be further modified.
- If the *ex* command resulted from a map expansion, all characters from that map expansion shall be discarded, except as otherwise specified by the **map** command.
- Otherwise, if the *ex* command resulted from the processing of an *EXINIT* environment variable, a **.exrc** file, a **:source** command, a **-c** option, or a **+command** specified to an *ex* **edit**, **ex**, **next**, or **visual** command, no further commands from the source of the commands shall be executed.
- Otherwise, if the *ex* command resulted from the execution of a buffer or a **global** or **v** command, no further commands caused by the execution of the buffer or the **global** or **v** command shall be executed.
- Otherwise, if the *ex* command was not terminated by a <newline>, all characters up to and including the next non-backslash-escaped <newline> shall be discarded.

### Input Editing in ex

The following symbol is used in this and the following sections to specify command actions:

*word*            In the POSIX locale, a word consists of a maximal sequence of letters, digits, and underscores, delimited at both ends by characters other than letters, digits, or underscores, or by the beginning or end of a line or the edit buffer.

When accepting input characters from the user, in either *ex* command mode or *ex* text input mode, *ex* shall enable canonical mode input processing, as defined in the System Interfaces volume of IEEE Std 1003.1-2001.

If in *ex* text input mode:

1. If the **number** edit option is set, *ex* shall prompt for input using the line number that would be assigned to the line if it is entered, in the format specified for the *ex* **number** command.
2. If the **autoindent** edit option is set, *ex* shall prompt for input using **autoindent** characters, as described by the **autoindent** edit option. **autoindent** characters shall follow the line number, if any.

If in *ex* command mode:

1. If the **prompt** edit option is set, input shall be prompted for using a single ' : ' character; otherwise, there shall be no prompt.

The input characters in the following sections shall have the following effects on the input line.

### Scroll

*Synopsis:*    *eof*

See the description of the *stty eof* character in *stty*.

If in *ex* command mode:

If the *eof* character is the first character entered on the line, the line shall be evaluated as if it contained two characters: a <control>-D and a <newline>.

Otherwise, the *eof* character shall have no special meaning.

14188 If in *ex* text input mode:

14189 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be  
 14190 modified so that a part of the next text input character will be displayed on the first column  
 14191 in the line after the previous **shiftwidth** edit option column boundary, and the user shall be  
 14192 prompted again for input for the same line.

14193 Otherwise, if the cursor follows a '0', which follows an **autoindent** character, and the '0'  
 14194 was the previous text input character, the '0' and all **autoindent** characters in the line shall  
 14195 be discarded, and the user shall be prompted again for input for the same line.

14196 Otherwise, if the cursor follows a '^', which follows an **autoindent** character, and the '^'  
 14197 was the previous text input character, the '^' and all **autoindent** characters in the line shall  
 14198 be discarded, and the user shall be prompted again for input for the same line. In addition,  
 14199 the **autoindent** level for the next input line shall be derived from the same line from which  
 14200 the **autoindent** level for the current input line was derived.

14201 Otherwise, if there are no **autoindent** or text input characters in the line, the *eof* character  
 14202 shall be discarded.

14203 Otherwise, the *eof* character shall have no special meaning.

14204 **<newline>**

14205 *Synopsis:* <newline>  
 14206 <control>-J

14207 If in *ex* command mode:

14208 Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for this  
 14209 purpose.

14210 If in *ex* text input mode:

14211 Terminate the current line. If there are no characters other than **autoindent** characters on the  
 14212 line, all characters on the line shall be discarded.

14213 Prompt for text input on a new line after the current line. If the **autoindent** edit option is set,  
 14214 an appropriate number of **autoindent** characters shall be added as a prefix to the line as  
 14215 described by the *ex* **autoindent** edit option.

14216 **<backslash>**

14217 *Synopsis:* <backslash>

14218 Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any  
 14219 special meaning that it may have to the editor during text input mode. The backslash character  
 14220 shall be retained and evaluated when the command line is parsed, or retained and included  
 14221 when the input text becomes part of the edit buffer.

|       |                                   |                                                                                                                                                |
|-------|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| 14222 | <b>&lt;control&gt;-V</b>          |                                                                                                                                                |
| 14223 | <i>Synopsis:</i>                  | <control>-V                                                                                                                                    |
| 14224 |                                   | Allow the entry of any subsequent character as a literal character, removing any special meaning                                               |
| 14225 |                                   | that it may have to the editor during text input mode. The <control>-V character shall be                                                      |
| 14226 |                                   | discarded before the command line is parsed or the input text becomes part of the edit buffer.                                                 |
| 14227 |                                   | If the “literal next” functionality is performed by the underlying system, it is implementation-                                               |
| 14228 |                                   | defined whether a character other than <control>-V performs this function.                                                                     |
| 14229 | <b>&lt;control&gt;-W</b>          |                                                                                                                                                |
| 14230 | <i>Synopsis:</i>                  | <control>-W                                                                                                                                    |
| 14231 |                                   | Discard the <control>-W, and the word previous to it in the input line, including any <blank>s                                                 |
| 14232 |                                   | following the word and preceding the <control>-W. If the “word erase” functionality is                                                         |
| 14233 |                                   | performed by the underlying system, it is implementation-defined whether a character other                                                     |
| 14234 |                                   | than <control>-W performs this function.                                                                                                       |
| 14235 | <b>Command Descriptions in ex</b> |                                                                                                                                                |
| 14236 |                                   | The following symbols are used in this section to represent command modifiers. Some of these                                                   |
| 14237 |                                   | modifiers can be omitted, in which case the specified defaults shall be used.                                                                  |
| 14238 | <i>1addr</i>                      | A single line address, given in any of the forms described in <b>Addressing in ex</b> (on                                                      |
| 14239 |                                   | page 361); the default shall be the current line ( ' . ' ), unless otherwise specified.                                                        |
| 14240 |                                   | If the line address is zero, it shall be an error, unless otherwise specified in the                                                           |
| 14241 |                                   | following command descriptions.                                                                                                                |
| 14242 |                                   | If the edit buffer is empty, and the address is specified with a command other than                                                            |
| 14243 |                                   | <b>=</b> , <b>append</b> , <b>insert</b> , <b>open</b> , <b>put</b> , <b>read</b> , or <b>visual</b> , or the address is not zero, it shall be |
| 14244 |                                   | an error.                                                                                                                                      |
| 14245 | <i>2addr</i>                      | Two addresses specifying an inclusive range of lines. If no addresses are specified,                                                           |
| 14246 |                                   | the default for <i>2addr</i> shall be the current line only ( " . , . " ), unless otherwise                                                    |
| 14247 |                                   | specified in the following command descriptions. If one address is specified, <i>2addr</i>                                                     |
| 14248 |                                   | shall specify that line only, unless otherwise specified in the following command                                                              |
| 14249 |                                   | descriptions.                                                                                                                                  |
| 14250 |                                   | It shall be an error if the first address is greater than the second address.                                                                  |
| 14251 |                                   | If the edit buffer is empty, and the two addresses are specified with a command                                                                |
| 14252 |                                   | other than the <b>!</b> , <b>write</b> , <b>wq</b> , or <b>xit</b> commands, or either address is not zero, it shall                           |
| 14253 |                                   | be an error.                                                                                                                                   |
| 14254 | <i>count</i>                      | A positive decimal number. If <i>count</i> is specified, it shall be equivalent to specifying                                                  |
| 14255 |                                   | an additional address to the command, unless otherwise specified by the following                                                              |
| 14256 |                                   | command descriptions. The additional address shall be equal to the last address                                                                |
| 14257 |                                   | specified to the command (either explicitly or by default) plus <i>count</i> −1.                                                               |
| 14258 |                                   | If this would result in an address greater than the last line of the edit buffer, it shall                                                     |
| 14259 |                                   | be corrected to equal the last line of the edit buffer.                                                                                        |
| 14260 | <i>flags</i>                      | One or more of the characters ' + ', ' − ', ' # ', ' p ', or ' l ' (ell). The flag characters                                                  |
| 14261 |                                   | can be <blank>-separated, and in any order or combination. The characters ' # ',                                                               |
| 14262 |                                   | ' p ', and ' l ' shall cause lines to be written in the format specified by the <b>print</b>                                                   |
| 14263 |                                   | command with the specified <i>flags</i> .                                                                                                      |

|       |               |                                                                                                                                                                                           |
|-------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 14264 |               | The lines to be written are as follows:                                                                                                                                                   |
| 14265 |               | 1. All edit buffer lines written during the execution of the <i>ex</i> <b>&amp;</b> , <b>~</b> , <b>list</b> , <b>number</b> ,                                                            |
| 14266 |               | <b>open</b> , <b>print</b> , <b>s</b> , <b>visual</b> , and <b>z</b> commands shall be written as specified by <i>flags</i> .                                                             |
| 14267 |               | 2. After the completion of an <i>ex</i> command with a flag as an argument, the                                                                                                           |
| 14268 |               | current line shall be written as specified by <i>flags</i> , unless the current line was                                                                                                  |
| 14269 |               | the last line written by the command.                                                                                                                                                     |
| 14270 |               | The characters '+' and '-' cause the value of the current line after the execution                                                                                                        |
| 14271 |               | of the <i>ex</i> command to be adjusted by the offset address as described in <b>Addressing</b>                                                                                           |
| 14272 |               | <b>in ex</b> (on page 361). This adjustment shall occur before the current line is written                                                                                                |
| 14273 |               | as described in 2. above.                                                                                                                                                                 |
| 14274 |               | The default for <i>flags</i> shall be none.                                                                                                                                               |
| 14275 | <i>buffer</i> | One of a number of named areas for holding text. The named buffers are specified                                                                                                          |
| 14276 |               | by the alphanumeric characters of the POSIX locale. There shall also be one                                                                                                               |
| 14277 |               | "unnamed" buffer. When no buffer is specified for editor commands that use a                                                                                                              |
| 14278 |               | buffer, the unnamed buffer shall be used. Commands that store text into buffers                                                                                                           |
| 14279 |               | shall store the text as it was before the command took effect, and shall store text                                                                                                       |
| 14280 |               | occurring earlier in the file before text occurring later in the file, regardless of how                                                                                                  |
| 14281 |               | the text region was specified. Commands that store text into buffers shall store the                                                                                                      |
| 14282 |               | text into the unnamed buffer as well as any specified buffer.                                                                                                                             |
| 14283 |               | In <i>ex</i> commands, buffer names are specified as the name by itself. In open or visual                                                                                                |
| 14284 |               | mode commands the name is preceded by a double quote (' ') character.                                                                                                                     |
| 14285 |               | If the specified buffer name is an uppercase character, and the buffer contents are                                                                                                       |
| 14286 |               | to be modified, the buffer shall be appended to rather than being overwritten. If                                                                                                         |
| 14287 |               | the buffer is not being modified, specifying the buffer name in lowercase and                                                                                                             |
| 14288 |               | uppercase shall have identical results.                                                                                                                                                   |
| 14289 |               | There shall also be buffers named by the numbers 1 through 9. In open and visual                                                                                                          |
| 14290 |               | mode, if a region of text including characters from more than a single line is being                                                                                                      |
| 14291 |               | modified by the <i>vi</i> <b>c</b> or <b>d</b> commands, the motion character associated with the <b>c</b> or                                                                             |
| 14292 |               | <b>d</b> commands specifies that the buffer text shall be in line mode, or the commands                                                                                                   |
| 14293 |               | <b>%</b> , <b>'</b> , <b>/</b> , <b>?</b> , <b>(</b> , <b>)</b> , <b>N</b> , <b>n</b> , <b>{</b> , or <b>}</b> are used to define a region of text for the <b>c</b> or <b>d</b> commands, |
| 14294 |               | the contents of buffers 1 through 8 shall be moved into the buffer named by the                                                                                                           |
| 14295 |               | next numerically greater value, the contents of buffer 9 shall be discarded, and the                                                                                                      |
| 14296 |               | region of text shall be copied into buffer 1. This shall be in addition to copying the                                                                                                    |
| 14297 |               | text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can                                                                                                         |
| 14298 |               | be specified as a source buffer for open and visual mode commands; however,                                                                                                               |
| 14299 |               | specifying a numeric buffer as the write target of an open or visual mode                                                                                                                 |
| 14300 |               | command shall have unspecified results.                                                                                                                                                   |
| 14301 |               | The text of each buffer shall have the characteristic of being in either line or                                                                                                          |
| 14302 |               | character mode. Appending text to a non-empty buffer shall set the mode to match                                                                                                          |
| 14303 |               | the characteristic of the text being appended. Appending text to a buffer shall                                                                                                           |
| 14304 |               | cause the creation of at least one additional line in the buffer. All text stored into                                                                                                    |
| 14305 |               | buffers by <i>ex</i> commands shall be in line mode. The <i>ex</i> commands that use buffers                                                                                              |
| 14306 |               | as the source of text specify individually how buffers of different modes are                                                                                                             |
| 14307 |               | handled. Each open or visual mode command that uses buffers for any purpose                                                                                                               |
| 14308 |               | specifies individually the mode of the text stored into the buffer and how buffers                                                                                                        |
| 14309 |               | of different modes are handled.                                                                                                                                                           |

14310 *file* Command text used to derive a pathname. The default shall be the current  
 14311 pathname, as defined previously, in which case, if no current pathname has yet  
 14312 been established it shall be an error, except where specifically noted in the  
 14313 individual command descriptions that follow. If the command text contains any of  
 14314 the characters '~', '{', '[', '\*', '?', '\$', '\', ' ', ' ', ' ', and '\', it shall be  
 14315 subjected to the process of “shell expansions”, as described below; if more than a  
 14316 single pathname results and the command expects only one, it shall be an error.

14317 The process of shell expansions in the editor shall be done as follows. The *ex* utility  
 14318 shall pass two arguments to the program named by the shell edit option; the first  
 14319 shall be **-c**, and the second shall be the string "echo" and the command text as a  
 14320 single argument. The standard output and standard error of that command shall  
 14321 replace the command text.

14322 **!** A character that can be appended to the command name to modify its operation,  
 14323 as detailed in the individual command descriptions. With the exception of the *ex*  
 14324 **read**, **write**, and **!** commands, the '!' character shall only act as a modifier if there  
 14325 are no <blank>s between it and the command name.

14326 *remembered search direction*

14327 The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in  
 14328 the edit buffer based on a remembered search direction, which is initially unset,  
 14329 and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* **/** and **?** commands.

## 14330 **Abbreviate**

14331 *Synopsis:* **ab**[*breviate*] [*lhs rhs*]

14332 If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

14333 Implementations may restrict the set of characters accepted in *lhs* or *rh*, except that printable  
 14334 characters and <blank>s shall not be restricted. Additional restrictions shall be implementation-  
 14335 defined.

14336 In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the  
 14337 character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be  
 14338 discarded.

14339 In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a  
 14340 <control>-V character is entered after a word character, a check shall be made for a set of  
 14341 characters matching *lhs*, in the text input entered during this command. If it is found, the effect  
 14342 shall be as if *rhs* was entered instead of *lhs*.

14343 The set of characters that are checked is defined as follows:

- 14344 1. If there are no characters inserted before the word and non-word or <ESC> characters that  
 14345 triggered the check, the set of characters shall consist of the word character.
- 14346 2. If the character inserted before the word and non-word or <ESC> characters that triggered  
 14347 the check is a word character, the set of characters shall consist of the characters inserted  
 14348 immediately before the triggering characters that are word characters, plus the triggering  
 14349 word character.
- 14350 3. If the character inserted before the word and non-word or <ESC> characters that triggered  
 14351 the check is not a word character, the set of characters shall consist of the characters that  
 14352 were inserted before the triggering characters that are neither <blank>s nor word  
 14353 characters, plus the triggering word character.

14354 It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate**  
 14355 commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the  
 14356 effect of the command shall be as if the replacement had not occurred.

14357 *Current line*: Unchanged.

14358 *Current column*: Unchanged.

## 14359 **Append**

14360 *Synopsis*:     [*laddr*] a[ppend] [!]

14361 Enter *ex* text input mode; the input text shall be placed after the specified line. If line zero is  
 14362 specified, the text shall be placed at the beginning of the edit buffer.

14363 This command shall be affected by the **number** and **autoindent** edit options; following the  
 14364 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
 14365 duration of this command only.

14366 *Current line*: Set to the last input line; if no lines were input, set to the specified line, or to the  
 14367 first line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.

14368 *Current column*: Set to non-<blank>.

## 14369 **Arguments**

14370 *Synopsis*:     ar[gs]

14371 Write the current argument list, with the current argument-list entry, if any, between '[' and  
 14372 ']' characters.

14373 *Current line*: Unchanged.

14374 *Current column*: Unchanged.

## 14375 **Change**

14376 *Synopsis*:     [*2addr*] c[hange] [!] [*count*]

14377 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall  
 14378 be copied into the unnamed buffer, which shall become a line mode buffer.

14379 This command shall be affected by the **number** and **autoindent** edit options; following the  
 14380 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
 14381 duration of this command only.

14382 *Current line*: Set to the last input line; if no lines were input, set to the line before the first  
 14383 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to  
 14384 zero if the edit buffer is empty.

14385 *Current column*: Set to non-<blank>.

**Change Directory**

Synopsis:    chd[ir] [!] [*directory*]  
               cd[!] [*directory*]

Change the current working directory to *directory*.

If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null and non-empty value, *directory* shall default to the value named in the *HOME* environment variable. If the *HOME* environment variable is empty or is undefined, the default value of *directory* is implementation-defined.

If no '!' is appended to the command name, and the edit buffer has been modified since the last complete write, and the current pathname does not begin with a '/', it shall be an error.

*Current line*: Unchanged.

*Current column*: Unchanged.

**Copy**

Synopsis:    [2*addr*] co[py] 1*addr* [*flags*]  
               [2*addr*] t 1*addr* [*flags*]

Copy the specified lines after the specified destination line; line zero specifies that the lines shall be placed at the beginning of the edit buffer.

*Current line*: Set to the last line copied.

*Current column*: Set to non-<blank>.

**Delete**

Synopsis:    [2*addr*] d[elete] [*buffer*] [*count*] [*flags*]

Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a line-mode buffer.

Flags can immediately follow the command name; see **Command Line Parsing in ex** (on page 362).

*Current line*: Set to the line following the deleted lines, or to the last line in the edit buffer if that line is past the end of the edit buffer, or to zero if the edit buffer is empty.

*Current column*: Set to non-<blank>.

**Edit**

Synopsis:    e[dit] [!] [+*command*] [*file*]  
               ex[!] [+*command*] [*file*]

If no '!' is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*, and set the current pathname to *file*. If *file* is not specified, replace the current contents of the edit buffer with the current contents of the file named by the current pathname. If for any reason the current contents of the file cannot be accessed, the edit buffer shall be empty.

The +*command* option shall be <blank>-delimited; <blank>s within +*command* can be escaped by preceding them with a backslash character. The +*command* shall be interpreted as an *ex* command immediately after the contents of the edit buffer have been replaced and the current



14426 line and column have been set.

14427 If the edit buffer is empty:

14428 *Current line*: Set to 0.

14429 *Current column*: Set to 1.

14430 Otherwise, if executed while in *ex* command mode or if the *+command* argument is specified:

14431 *Current line*: Set to the last line of the edit buffer.

14432 *Current column*: Set to non-<blank>.

14433 Otherwise, if *file* is omitted or results in the current pathname:

14434 *Current line*: Set to the first line of the edit buffer.

14435 *Current column*: Set to non-<blank>.

14436 Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if

14437 the file was previously edited, the line and column may be set as follows:

14438 *Current line*: Set to the last value held when that file was last edited. If this value is not a valid

14439 line in the new edit buffer, set to the first line of the edit buffer.

14440 *Current column*: If the current line was set to the last value held when the file was last edited, set

14441 to the last value held when the file was last edited. Otherwise, or if the last value is not a valid

14442 column in the new edit buffer, set to non-<blank>.

14443 Otherwise:

14444 *Current line*: Set to the first line of the edit buffer.

14445 *Current column*: Set to non-<blank>.

14446 **File**

14447 *Synopsis:*     f [*ile*] [*file*]

14448 If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and

14449 the current pathname shall be set to *file*.

14450 Write an informational message. If the file has a current pathname, it shall be included in this

14451 message; otherwise, the message shall indicate that there is no current pathname. If the edit

14452 buffer contains lines, the current line number and the number of lines in the edit buffer shall be

14453 included in this message; otherwise, the message shall indicate that the edit buffer is empty. If

14454 the edit buffer has been modified since the last complete write, this fact shall be included in this

14455 message. If the **readonly** edit option is set, this fact shall be included in this message. The

14456 message may contain other unspecified information.

14457 *Current line*: Unchanged.

14458 *Current column*: Unchanged.

**Global**

Synopsis:     [2addr] g[lobal] /pattern/ [commands]  
               [2addr] v /pattern/ [commands]

The optional '!' character after the **global** command shall be the same as executing the **v** command.

If *pattern* is empty (for example, "//") or not specified, the last regular expression used in the editor command shall be used as the *pattern*. The *pattern* can be delimited by slashes (shown in the Synopsis), as well as any non-alphanumeric or non-<blank> other than backslash, vertical line, double quote, or <newline>.

If no lines are specified, the lines shall default to the entire file.

The **global** and **v** commands are logically two-pass operations. First, mark the lines within the specified lines for which the line excluding the terminating <newline> matches (**global**) or does not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by *commands*, with the current line ('.') set to each marked line. If an error occurs during this process, or the contents of the edit buffer are replaced (for example, by the *ex* **:edit** command) an error message shall be written and no more commands resulting from the execution of this command shall be processed.

Multiple *ex* commands can be specified by entering multiple commands on a single line using a vertical line to delimit them, or one per line, by escaping each <newline> with a backslash.

If no commands are specified:

1. If in *ex* command mode, it shall be as if the **print** command were specified.
2. Otherwise, no command shall be executed.

For the **append**, **change**, and **insert** commands, the input text shall be included as part of the command, and the terminating period can be omitted if the command ends the list of commands. The **open** and **visual** commands can be specified as one of the commands, in which case each marked line shall cause the editor to enter open or visual mode. If open or visual mode is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open or visual mode reentered, until the list of marked lines is exhausted.

The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted by commands executed for lines occurring earlier in the file than the marked lines. In this case, no commands shall be executed for the deleted lines.

If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v** command.

*Current line*: If no commands executed, set to the last marked line. Otherwise, as specified for the executed *ex* commands.

*Current column*: If no commands are executed, set to non-<blank>; otherwise, as specified for the individual *ex* commands.

14497      **Insert**14498      *Synopsis:*      [*laddr*] i[nsert] [!]14499      Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero  
14500      or 1, the text shall be placed at the beginning of the edit buffer.14501      This command shall be affected by the **number** and **autoindent** edit options; following the  
14502      command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
14503      duration of this command only.14504      *Current line:* Set to the last input line; if no lines were input, set to the line before the specified  
14505      line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero  
14506      if the edit buffer is empty.14507      *Current column:* Set to non-<blank>.14508      **Join**14509      *Synopsis:*      [*2addr*] j[oin] [!] [*count*] [*flags*]14510      If *count* is specified:14511          If no address was specified, the **join** command shall behave as if *2addr* were the current line  
14512          and the current line plus *count* (.,. + *count*).14513          If one address was specified, the **join** command shall behave as if *2addr* were the specified  
14514          address and the specified address plus *count* (*addr*,*addr* + *count*).14515          If two addresses were specified, the **join** command shall behave as if an additional address,  
14516          equal to the last address plus *count* - 1 (*addr1*,*addr2*,*addr2* + *count* - 1), was specified.14517          If this would result in a second address greater than the last line of the edit buffer, it shall be  
14518          corrected to be equal to the last line of the edit buffer.14519      If no *count* is specified:14520          If no address was specified, the **join** command shall behave as if *2addr* were the current line  
14521          and the next line (.,. + 1).14522          If one address was specified, the **join** command shall behave as if *2addr* were the specified  
14523          address and the next line (*addr*,*addr* + 1).14524      Join the text from the specified lines together into a single line, which shall replace the specified  
14525      lines.14526      If a '!' character is appended to the command name, the **join** shall be without modification of  
14527      any line, independent of the current locale.14528      Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for  
14529      each subsequent line, proceed as follows:

- 14530          1. Discard leading <space>s from the line to be joined.
- 14531          2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
- 14532          3. If the current line ends in a <blank>, or the first character of the line to be joined is a ' ) ' character, join the lines without further modification.
- 14533          4. If the last character of the current line is a ' . ' , join the lines with two <space>s between them.
- 14534          5.
- 14535          6.

14536 5. Otherwise, join the lines with a single <space> between them.

14537 *Current line*: Set to the first line specified.

14538 *Current column*: Set to non-<blank>.

## 14539 **List**

14540 *Synopsis*: [2addr] l[ist] [count] [flags]

14541 This command shall be equivalent to the *ex* command:

14542 [2addr] p[rint] [count] l[flags]

14543 See **Print** (on page 380).

## 14544 **Map**

14545 *Synopsis*: map[!] [lhs rhs]

14546 If *lhs* and *rhs* are not specified:

- 14547 1. If '!' is specified, write the current list of text input mode maps.
- 14548 2. Otherwise, write the current list of command mode maps.
- 14549 3. Do nothing more.

14550 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable  
 14551 characters and <blank>s shall not be restricted. Additional restrictions shall be implementation-  
 14552 defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in which case the  
 14553 character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be  
 14554 discarded.

14555 If the character '!' is appended to the **map** command name, the mapping shall be effective  
 14556 during open or visual text input mode rather than **open** or **visual** command mode. This allows  
 14557 *lhs* to have two different **map** definitions at the same time: one for command mode and one for  
 14558 text input mode.

14559 For command mode mappings:

14560 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as part  
 14561 of the arguments to the command), the action shall be as if the corresponding *rhs* had been  
 14562 entered.

14563 If any character in the command, other than the first, is escaped using a <control>-V  
 14564 character, that character shall not be part of a match to an *lhs*.

14565 It is unspecified whether implementations shall support **map** commands where the *lhs* is  
 14566 more than a single character in length, where the first character of the *lhs* is printable.

14567 If *lhs* contains more than one character and the first character is '#', followed by a sequence  
 14568 of digits corresponding to a numbered function key, then when this function key is typed it  
 14569 shall be mapped to *rhs*. Characters other than digits following a '#' character also represent  
 14570 the function key named by the characters in the *lhs* following the '#' and may be mapped to  
 14571 *rhs*. It is unspecified how function keys are named or what function keys are supported.

14572 For text input mode mappings:

- 14573 When the *lhs* is entered as any part of text entered in open or visual text input modes, the  
14574 action shall be as if the corresponding *rhs* had been entered.
- 14575 If any character in the input text is escaped using a <control>-V character, that character shall  
14576 not be part of a match to an *lhs*.
- 14577 It is unspecified whether the *lhs* text entered for subsequent **map** or **unmap** commands is  
14578 replaced with the *rhs* text for the purposes of the screen display; regardless of whether or not  
14579 the display appears as if the corresponding *rhs* text was entered, the effect of the command  
14580 shall be as if the *lhs* text was entered.
- 14581 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,  
14582 possibly matching characters before treating the already entered characters as not matching the  
14583 *lhs*.
- 14584 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the  
14585 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those  
14586 characters shall not be remapped.
- 14587 On block-mode terminals, the mapping need not occur immediately (for example, it may occur  
14588 after the terminal transmits a group of characters to the system), but it shall achieve the same  
14589 results as if it occurred immediately.
- 14590 *Current line*: Unchanged.
- 14591 *Current column*: Unchanged.
- 14592 **Mark**
- 14593 *Synopsis:*     [*laddr*] ma[*rk*] *character*  
14594               [*laddr*] k *character*
- 14595 Implementations shall support *character* values of a single lowercase letter of the POSIX locale  
14596 and the characters ' ' and ' '; support of other characters is implementation-defined.
- 14597 If executing the vi **m** command, set the specified mark to the current line and 1-based numbered  
14598 character referenced by the current column, if any; otherwise, column position 1.
- 14599 Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank>  
14600 non-<newline> in the line, if any; otherwise, the last non-<newline> in the line, if any; otherwise,  
14601 column position 1.
- 14602 The mark shall remain associated with the line until the mark is reset or the line is deleted. If a  
14603 deleted line is restored by a subsequent **undo** command, any marks previously associated with  
14604 the line, which have not been reset, shall be restored as well. Any use of a mark not associated  
14605 with a current line in the edit buffer shall be an error.
- 14606 The marks ' and ' shall be set as described previously, immediately before the following events  
14607 occur in the editor:
- 14608     1. The use of ' \$ ' as an *ex* address
  - 14609     2. The use of a positive decimal number as an *ex* address
  - 14610     3. The use of a search command as an *ex* address
  - 14611     4. The use of a mark reference as an *ex* address
  - 14612     5. The use of the following open and visual mode commands: <control>-], %, (, ), [, ], {, }
  - 14613     6. The use of the following open and visual mode commands: ', **G**, **H**, **L**, **M**, **z** if the current  
14614 line will change as a result of the command

7. The use of the open and visual mode commands: */*, *?*, **N**, *'*, **n** if the current line or column will change as a result of the command

8. The use of the ex mode commands: **z**, **undo**, **global**, **v**

For rules 1., 2., 3., and 4., the *'* and *'* marks shall not be set if the ex command is parsed as specified by rule 6.a. in **Command Line Parsing in ex** (on page 362).

For rules 5., 6., and 7., the *'* and *'* marks shall not be set if the commands are used as motion commands in open and visual mode.

For rules 1., 2., 3., 4., 5., 6., 7., and 8., the *'* and *'* marks shall not be set if the command fails.

The *'* and *'* marks shall be set as described previously, each time the contents of the edit buffer are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex** mode and the edit buffer is not empty, before any commands or movements (including commands or movements specified by the **-c** or **-t** options or the **+command** argument) are executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the **vi m** command; otherwise, as if executing the ex **mark** command.

When changing from **ex** mode to open or visual mode, if the *'* and *'* marks are not already set, the *'* and *'* marks shall be set as described previously.

*Current line*: Unchanged.

*Current column*: Unchanged.

## Move

*Synopsis*:     [2addr] m[ove] 1addr [flags]

Move the specified lines after the specified destination line. A destination of line zero specifies that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the destination line is within the range of lines to be moved.

*Current line*: Set to the last of the moved lines.

*Current column*: Set to non-<blank>.

## Next

*Synopsis*:     n[ext] [!] [+command] [file ...]

If no *' ! '* is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

If one or more files is specified:

1. Set the argument list to the specified filenames.
2. Set the current argument list reference to be the first entry in the argument list.
3. Set the current pathname to the first filename specified.

Otherwise:

1. It shall be an error if there are no more filenames in the argument list after the filename currently referenced.
2. Set the current pathname and the current argument list reference to the filename after the filename currently referenced in the argument list.

14654 Replace the contents of the edit buffer with the contents of the file named by the current  
 14655 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be  
 14656 empty.

14657 This command shall be affected by the **autowrite** and **writeany** edit options.

14658 The *+command* option shall be <blank>-delimited; <blank>s can be escaped by preceding them  
 14659 with a backslash character. The *+command* shall be interpreted as an *ex* command immediately  
 14660 after the contents of the edit buffer have been replaced and the current line and column have  
 14661 been set.

14662 *Current line*: Set as described for the **edit** command.

14663 *Current column*: Set as described for the **edit** command.

## 14664 **Number**

14665 *Synopsis:*     [2addr] nu[mber] [count] [flags]  
 14666                 [2addr] #[count] [flags]

14667 These commands shall be equivalent to the *ex* command:

14668     [2addr] p[rint] [count] #[flags]

14669 See **Print** (on page 380).

## 14670 **Open**

14671 *Synopsis:*     [1addr] o[pen] /pattern/ [flags]

14672 This command need not be supported on block-mode terminals or terminals with insufficient  
 14673 capabilities. If standard input, standard output, or standard error are not terminal devices, the  
 14674 results are unspecified.

14675 Enter open mode.

14676 The trailing delimiter can be omitted from *pattern* at the end of the command line. If *pattern* is  
 14677 empty (for example, " // ") or not specified, the last regular expression used in the editor shall be  
 14678 used as the pattern. The pattern can be delimited by slashes (shown in the Synopsis), as well as  
 14679 any alphanumeric, or non-<blank> other than backslash, vertical line, double quote, or  
 14680 <newline>.

14681 *Current line*: Set to the specified line.

14682 *Current column*: Set to non-<blank>.

## 14683 **Preserve**

14684 *Synopsis:*     pre[serve]

14685 Save the edit buffer in a form that can later be recovered by using the **-r** option or by using the *ex*  
 14686 **recover** command. After the file has been preserved, a mail message shall be sent to the user.  
 14687 This message shall be readable by invoking the *mailx* utility. The message shall contain the name  
 14688 of the file, the time of preservation, and an *ex* command that could be used to recover the file.  
 14689 Additional information may be included in the mail message.

14690 *Current line*: Unchanged.

14691 *Current column*: Unchanged.

**Print**

*Synopsis:*     [2addr] p[rint] [count] [flags]

Write the addressed lines. The behavior is unspecified if the number of columns on the display is less than the number of columns required to write any single character in the lines being written.

Non-printable characters, except for the <tab>, shall be written as implementation-defined multi-character sequences.

If the # flag is specified or the **number** edit option is set, each line shall be preceded by its line number in the following format:

"%6dΔΔ", <line number>

If the **l** flag is specified or the **list** edit option is set:

1. The characters listed in the Base Definitions volume of IEEE Std 1003.1-2001, Table 5-1, Escape Sequences and Associated Actions shall be written as the corresponding escape sequence.
2. Non-printable characters not in the Base Definitions volume of IEEE Std 1003.1-2001, Table 5-1, Escape Sequences and Associated Actions shall be written as one three-digit octal number (with a preceding backslash) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than 9 bits, the format used for non-printable characters is implementation-defined.
3. The end of each line shall be marked with a '\$', and literal '\$' characters within the line shall be written with a preceding backslash.

Long lines shall be folded; the length at which folding occurs is unspecified, but should be appropriate for the output terminal, considering the number of columns of the terminal.

If a line is folded, and the **l** flag is not specified and the **list** edit option is not set, it is unspecified whether a multi-column character at the folding position is separated; it shall not be discarded.

*Current line:* Set to the last written line.

*Current column:* Unchanged if the current line is unchanged; otherwise, set to non-<blank>.

**Put**

*Synopsis:*     [laddr] pu[t] [buffer]

Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

*Current line:* Set to the last line entered into the edit buffer.

*Current column:* Set to non-<blank>.

**Quit**

*Synopsis:*     q[uit] [!]

If no '!' is appended to the command name:

1. If the edit buffer has been modified since the last complete write, it shall be an error.
2. If there are filenames in the argument list after the filename currently referenced, and the last command was not a **quit**, **wq**, **xit**, or **ZZ** (see **Exit** (on page 1022)) command, it shall be an error.



14732 Otherwise, terminate the editing session.

## 14733 **Read**

14734 *Synopsis:*     `[laddr] r[ead] [!] [file]`

14735 If '!' is not the first non-`<blank>` to follow the command name, a copy of the specified file shall  
 14736 be appended into the edit buffer after the specified line; line zero specifies that the copy shall be  
 14737 placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If  
 14738 no *file* is named, the current pathname shall be the default. If there is no current pathname, then  
 14739 *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be  
 14740 an error. Specifying a *file* that is not of type regular shall have unspecified results.

14741 Otherwise, if *file* is preceded by '!', the rest of the line after the '!' shall have '%', '#', and  
 14742 '!' characters expanded as described in **Command Line Parsing in ex** (on page 362).

14743 The *ex* utility shall then pass two arguments to the program named by the shell edit option; the  
 14744 first shall be `-c` and the second shall be the expanded arguments to the **read** command as a  
 14745 single argument. The standard input of the program shall be set to the standard input of the *ex*  
 14746 program when it was invoked. The standard error and standard output of the program shall be  
 14747 appended into the edit buffer after the specified line.

14748 Each line in the copied file or program output (as delimited by `<newline>`s or the end of the file  
 14749 or output if it is not immediately preceded by a `<newline>`), shall be a separate line in the edit  
 14750 buffer. Any occurrences of `<carriage-return>` and `<newline>` pairs in the output shall be treated  
 14751 as single `<newline>`s.

14752 The special meaning of the '!' following the **read** command can be overridden by escaping it  
 14753 with a backslash character.

14754 *Current line:* If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual  
 14755 mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into  
 14756 the edit buffer.

14757 *Current column:* Set to non-`<blank>`.

## 14758 **Recover**

14759 *Synopsis:*     `rec[over] [!] file`

14760 If no '!' is appended to the command name, and the edit buffer has been modified since the  
 14761 last complete write, it shall be an error.

14762 If no *file* operand is specified, then the current pathname shall be used. If there is no current  
 14763 pathname or *file* operand, it shall be an error.

14764 If no recovery information has previously been saved about *file*, the **recover** command shall  
 14765 behave identically to the **edit** command, and an informational message to this effect shall be  
 14766 written.

14767 Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer  
 14768 with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the  
 14769 one most recently saved shall be recovered, and an informational message that there are  
 14770 previous versions of the file that can be recovered shall be written. The editor shall behave as if  
 14771 the contents of the edit buffer have already been modified.

14772 *Current file:* Set as described for the **edit** command.

14773 *Current column:* Set as described for the **edit** command.

**Rewind**

*Synopsis:*     rew[ind] [!]

If no '!' is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

If the argument list is empty, it shall be an error.

The current argument list reference and the current pathname shall be set to the first filename in the argument list.

Replace the contents of the edit buffer with the contents of the file named by the current pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

This command shall be affected by the **autowrite** and **writeany** edit options.

*Current line:* Set as described for the **edit** command.

*Current column:* Set as described for the **edit** command.

**Set**

*Synopsis:*     se[t] [option[=*value*]] ... [nooption ...] [option? ...] [all]

When no arguments are specified, write the value of the **term** edit option and those options whose values have been changed from the default settings; when the argument *all* is specified, write all of the option values.

Giving an option name followed by the character '?' shall cause the current value of that option to be written. The '?' can be separated from the option name by zero or more <blank>s. The '?' shall be necessary only for Boolean valued options. Boolean options can be given values by the form **set option** to turn them on or **set nooption** to turn them off; string and numeric options can be assigned by the form **set option=value**. Any <blank>s in strings can be included as is by preceding each <blank> with an escaping backslash. More than one option can be set or listed by a single set command by specifying multiple arguments, each separated from the next by one or more <blank>s.

See **Edit Options in ex** (on page 392) for details about specific options.

*Current line:* Unchanged.

*Current column:* Unchanged.

**Shell**

*Synopsis:*     sh[ell]

Invoke the program named in the **shell** edit option with the single argument **-i** (interactive mode). Editing shall be resumed when the program exits.

*Current line:* Unchanged.

*Current column:* Unchanged.

**Source**

*Synopsis:*     `so[urce] file`

Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.

*Current line:* As specified for the individual *ex* commands.

*Current column:* As specified for the individual *ex* commands.

**Substitute**

*Synopsis:*     `[2addr] s[substitute] [/pattern/repl/[options] [count] [flags]]`  
                   `[2addr] &[options] [count] [flags]`  
                   `[2addr] ~[options] [count] [flags]`

Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See **Regular Expressions in ex** (on page 391) and **Replacement Strings in ex** (on page 391).) Any non-alphabetic, non-`<blank>` delimiter other than `'\'`, `'|'`, double quote, or `<newline>` can be used instead of `'/'`. Backslash characters can be used to escape delimiters, backslash characters, and other special characters.

The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If both *pattern* and *repl* are not specified or are empty (for example, `"/"/`), the last **s** command shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be replaced by nothing. If the entire replacement pattern is `'%'`, the last replacement pattern to an **s** command shall be used.

Entering a `<carriage-return>` in *repl* (which requires an escaping backslash in *ex* mode and an escaping `<control>-V` in open or *vi* mode) shall split the line at that point, creating a new line in the edit buffer. The `<carriage-return>` shall be discarded.

If *options* includes the letter `'g'` (**global**), all non-overlapping instances of the pattern in the line shall be replaced.

If *options* includes the letter `'c'` (**confirm**), then before each substitution the line shall be written; the written line shall reflect all previous substitutions. On the following line, `<space>s` shall be written beneath the characters from the line that are before the *pattern* to be replaced, and `'^'` characters written beneath the characters included in the *pattern* to be replaced. The *ex* utility shall then wait for a response from the user. An affirmative response shall cause the substitution to be done, while any other input shall not make the substitution. An affirmative response shall consist of a line with the affirmative response (as defined by the current locale) at the beginning of the line. This line shall be subject to editing in the same way as the *ex* command line.

If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the user shall be preserved in the edit buffer after the interrupt.

If the remembered search direction is not set, the **s** command shall set it to forward.

In the second Synopsis, the **&** command shall repeat the previous substitution, as if the **&** command were replaced by:

`s/pattern/repl/`

where *pattern* and *repl* are as specified in the previous **s**, **&**, or **~** command.

In the third Synopsis, the **~** command shall repeat the previous substitution, as if the `'~'` were replaced by:

14852 *s/pattern/repl/*

14853 where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from  
14854 the previous substitution (including **&** and **~**) command.

14855 These commands shall be affected by the *LC\_MESSAGES* environment variable.

14856 *Current line*: Set to the last line in which a substitution occurred, or, unchanged if no  
14857 substitution occurred.

14858 *Current column*: Set to non-<blank>.

## 14859 **Suspend**

14860 *Synopsis:*     su [spend] [!]  
14861               st [op] [!]

14862 Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the  
14863 SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell  
14864 (see the description of *set -m*).

14865 These commands shall be affected by the **autowrite** and **writeany** edit options.

14866 The current **susp** character (see *stty*) shall be equivalent to the **suspend** command.

## 14867 **Tag**

14868 *Synopsis:*     ta [g] [!] *tagstring*

14869 The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see  
14870 *ctags*) description.

14871 The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in  
14872 the order they are specified, until a reference to *tagstring* is found. Files shall be searched from  
14873 beginning to end. If no reference is found, it shall be an error and an error message to this effect  
14874 shall be written. If the reference is not found, or if an error occurs while processing a file referred  
14875 to in the **tag** edit option, it shall be an error, and an error message shall be written at the first  
14876 occurrence of such an error.

14877 Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression  
14878 used in the editor; for example, for the purposes of the **s** command.

14879 If the *tagstring* is in a file with a different name than the current pathname, set the current  
14880 pathname to the name of that file, and replace the contents of the edit buffer with the contents of  
14881 that file. In this case, if no '!' is appended to the command name, and the edit buffer has been  
14882 modified since the last complete write, it shall be an error, unless the file is successfully written  
14883 as specified by the **autowrite** option.

14884 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

14885 *Current line*: If the tags file contained a line number, set to that line number. If the line number is  
14886 larger than the last line in the edit buffer, an error message shall be written and the current line  
14887 shall be set as specified for the **edit** command.

14888 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no  
14889 matching pattern is found, an error message shall be written and the current line shall be set as  
14890 specified for the **edit** command.

14891 *Current column*: If the tags file contained a line-number reference and that line-number was not  
14892 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern  
14893 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

**Unabbreviate**

*Synopsis:*     una[bbrev] lhs

If *lhs* is not an entry in the current list of abbreviations (see **Abbreviate** (on page 370)), it shall be an error. Otherwise, delete *lhs* from the list of abbreviations.

*Current line:* Unchanged.

*Current column:* Unchanged.

**Undo**

*Synopsis:*     u[ndo]

Reverse the changes made by the last command that modified the contents of the edit buffer, including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands resulting from buffer executions and mapped character expansions, are considered single commands.

If no action that can be undone preceded the **undo** command, it shall be an error.

If the **undo** command restores lines that were marked, the mark shall also be restored unless it was reset subsequent to the deletion of the lines.

*Current line:*

1. If lines are added or changed in the file, set to the first line added or changed.
2. Set to the line before the first line deleted, if it exists.
3. Set to 1 if the edit buffer is not empty.
4. Set to zero.

*Current column:* Set to non-<blank>.

**Unmap**

*Synopsis:*     unm[ap] [!] lhs

If '!' is appended to the command name, and if *lhs* is not an entry in the list of text input mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map definitions.

If no '!' is appended to the command name, and if *lhs* is not an entry in the list of command mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode map definitions.

*Current line:* Unchanged.

*Current column:* Unchanged.

**Version**

*Synopsis:*     ve[rsion]

Write a message containing version information for the editor. The format of the message is unspecified.

*Current line:* Unchanged.

*Current column:* Unchanged.

**Visual**

*Synopsis:*     [*laddr*] vi[*sual*] [*type*] [*count*] [*flags*]

If *ex* is currently in open or visual mode, the Synopsis and behavior of the visual command shall be the same as the **edit** command, as specified by **Edit** (on page 372).

Otherwise, this command need not be supported on block-mode terminals or terminals with insufficient capabilities. If standard input, standard output, or standard error are not terminal devices, the results are unspecified.

If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in **window** (on page 398)). If the '^' type character was also specified, the **window** edit option shall be set before being used by the type character.

Enter visual mode. If *type* is not specified, it shall be as if a *type* of '+' was specified. The *type* shall cause the following effects:

- + Place the beginning of the specified line at the top of the display.
- Place the end of the specified line at the bottom of the display.
- . Place the beginning of the specified line in the middle of the display.
- ^ If the specified line is less than or equal to the value of the **window** edit option, set the line to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place the beginning of this line as close to the bottom of the displayed lines as possible, while still displaying the value of the **window** edit option number of lines.

*Current line:* Set to the specified line.

*Current column:* Set to non-<blank>.

**Write**

*Synopsis:*     [*2addr*] w[*rite*] [!] [>>] [*file*]  
                   [*2addr*] w[*rite*] [!] [*file*]  
                   [*2addr*] wq[!] [>>] [*file*]

If no lines are specified, the lines shall default to the entire file.

The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!** shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the **quit** shall not be attempted.

If the command name is not followed by one or more <blank>s, or *file* is not preceded by a '!' character, the **write** shall be to a file.

1. If the >> argument is specified, and the file already exists, the lines shall be appended to the file instead of replacing its contents. If the >> argument is specified, and the file does not already exist, it is unspecified whether the write shall proceed as if the >> argument had not been specified or if the write shall fail.
2. If the **readonly** edit option is set (see **readonly** (on page 395)), the **write** shall fail.
3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.
4. If *file* is not specified, the current pathname shall be used. If there is no current pathname, the **write** command shall fail.
5. If the current pathname is used, and the current pathname has been changed by the **file** or **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,

14972 subsequent **writes** shall not fail for this reason (unless the current pathname is changed  
14973 again).

14974 6. If the whole edit buffer is not being written, and the file to be written exists, the **write** shall  
14975 fail.

14976 For rules 1., 2., 4., and 5., the **write** can be forced by appending the character **'!'** to the  
14977 command name.

14978 For rules 2., 4., and 5., the **write** can be forced by setting the **writeany** edit option.

14979 Additional, implementation-defined tests may cause the **write** to fail.

14980 If the edit buffer is empty, a file without any contents shall be written.

14981 An informational message shall be written noting the number of lines and bytes written.

14982 Otherwise, if the command is followed by one or more <blank>s, and the file is preceded by  
14983 **'!'**, the rest of the line after the **'!'** shall have **'%'**, **'#'**, and **'!'** characters expanded as  
14984 described in **Command Line Parsing in ex** (on page 362).

14985 The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the  
14986 first shall be **-c** and the second shall be the expanded arguments to the **write** command as a  
14987 single argument. The specified lines shall be written to the standard input of the command. The  
14988 standard error and standard output of the program, if any, shall be written as described for the  
14989 **print** command. If the last character in that output is not a <newline>, a <newline> shall be  
14990 written at the end of the output.

14991 The special meaning of the **'!'** following the **write** command can be overridden by escaping it  
14992 with a backslash character.

14993 *Current line:* Unchanged.

14994 *Current column:* Unchanged.

## 14995 **Write and Exit**

14996 *Synopsis:*     [2addr] x[it] [!] [file]

14997 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to  
14998 the **quit** command, or if a **'!'** is appended to the command name, to **quit!**.

14999 Otherwise, **xit** shall be equivalent to the **wq** command, or if a **'!'** is appended to the command  
15000 name, to **wq!**.

15001 *Current line:* Unchanged.

15002 *Current column:* Unchanged.

## 15003 **Yank**

15004 *Synopsis:*     [2addr] ya[nk] [buffer] [count]

15005 Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall  
15006 become a line-mode buffer.

15007 *Current line:* Unchanged.

15008 *Current column:* Unchanged.

**Adjust Window**

*Synopsis:*     [*laddr*] *z*[!][*type* ...][*count*][*flags*]

If no line is specified, the current line shall be the default; if *type* is omitted as well, the current line value shall first be incremented by 1. If incrementing the current line would cause it to be greater than the last line in the edit buffer, it shall be an error.

If there are <blank>s between the *type* argument and the preceding **z** command name or optional '!' character, it shall be an error.

If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in **window** (on page 398)). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit option, or if ! was specified, the number of lines in the display minus 1.

If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise, *count* lines starting with the line specified by the *type* argument shall be written.

The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

- The specified line shall be decremented by the following value:

$((\text{number of ``-'' characters}) \times \text{count}) - 1$

If the calculation would result in a number less than 1, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

- + The specified line shall be incremented by the following value:

$((\text{number of ``+'' characters}) - 1) \times \text{count} + 1$

If the calculation would result in a number greater than the last line in the edit buffer, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

- =, . If more than a single '.' or '=' is specified, it shall be an error. The following steps shall be taken:

1. If *count* is zero, nothing shall be written.
2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count* or '!' was specified, *N* shall be:

$(\text{count} - 1) / 2$

Otherwise, *N* shall be:

$(\text{count} - 3) / 2$

If *N* is a number less than 3, no lines shall be written.

3. If '=' was specified as the type character, write a line consisting of the smaller of the number of columns in the display divided by two, or 40 '-' characters.
4. Write the current line.
5. Repeat step 3.
6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall be defined as in step 2. If *N* is a number less than 3, no lines shall be written. If *count* is less than 3, no lines shall be written.



15048        ^    The specified line shall be decremented by the following value:

15049         $((\text{number of ``^'' characters}) + 1) \times \text{count} - 1$

15050        If the calculation would result in a number less than 1, it shall be an error. Write lines from

15051        the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit

15052        buffer has been written.

15053        *Current line*: Set to the last line written, unless the type is =, in which case, set to the specified

15054        line.

15055        *Current column*: Set to non-<blank>.

15056        **Escape**

15057        *Synopsis*:        ! *command*

15058                       [*addr*] ! *command*

15059        The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as

15060        described in **Command Line Parsing in ex** (on page 362). If the expansion causes the text of the

15061        line to change, it shall be redisplayed, preceded by a single '!' character.

15062        The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two

15063        arguments to the program; the first shall be **-c**, and the second shall be the expanded arguments

15064        to the ! command as a single argument.

15065        If no lines are specified, the standard input, standard output, and standard error of the program

15066        shall be set to the standard input, standard output, and standard error of the *ex* program when it

15067        was invoked. In addition, a warning message shall be written if the edit buffer has been

15068        modified since the last complete write, and the **warn** edit option is set.

15069        If lines are specified, they shall be passed to the program as standard input, and the standard

15070        output and standard error of the program shall replace those lines in the edit buffer. Each line in

15071        the program output (as delimited by <newline>s or the end of the output if it is not immediately

15072        preceded by a <newline>), shall be a separate line in the edit buffer. Any occurrences of

15073        <carriage-return> and <newline> pairs in the output shall be treated as single <newline>s. The

15074        specified lines shall be copied into the unnamed buffer before they are replaced, and the

15075        unnamed buffer shall become a line-mode buffer.

15076        If in *ex* mode, a single '!' character shall be written when the program completes.

15077        This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this

15078        command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this

15079        command shall be affected by the **autoprint** edit option.

15080        *Current line*:

15081               1. If no lines are specified, unchanged.

15082               2. Otherwise, set to the last line read in, if any lines are read in.

15083               3. Otherwise, set to the line before the first line of the lines specified, if that line exists.

15084               4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.

15085               5. Otherwise, set to zero.

15086        *Current column*: If no lines are specified, unchanged. Otherwise, set to non-<blank>.

**Shift Left**

*Synopsis:*     [2addr] <[< ...] [count] [flags]

Shift the specified lines to the start of the line; the number of column positions to be shifted shall be the number of command characters times the value of the **shiftwidth** edit option. Only leading <blank>s shall be deleted or changed into other <blank>s in shifting; other characters shall not be affected.

Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode buffer.

This command shall be affected by the **autoprint** edit option.

*Current line:* Set to the last line in the lines specified.

*Current column:* Set to non-<blank>.

**Shift Right**

*Synopsis:*     [2addr] >[> ...] [count] [flags]

Shift the specified lines away from the start of the line; the number of column positions to be shifted shall be the number of command characters times the value of the **shiftwidth** edit option. The shift shall be accomplished by adding <blank>s as a prefix to the line or changing leading <blank>s into other <blank>s. Empty lines shall not be changed.

Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode buffer.

This command shall be affected by the **autoprint** edit option.

*Current line:* Set to the last line in the lines specified.

*Current column:* Set to non-<blank>.

**<control>-D**

*Synopsis:*     <control>-D

Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the number of lines after the current line in the edit buffer. If the current line is the last line of the edit buffer it shall be an error.

*Current line:* Set to the last line written.

*Current column:* Set to non-<blank>.

**Write Line Number**

*Synopsis:*     [1addr] = [flags]

If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of the specified line.

*Current line:* Unchanged.

*Current column:* Unchanged.

**Execute**

Synopsis:     [2addr] @ *buffer*  
               [2addr] \* *buffer*

If no buffer is specified or is specified as '@' or '\*', the last buffer executed shall be used. If no previous buffer has been executed, it shall be an error.

For each line specified by the addresses, set the current line ('.') to the specified line, and execute the contents of the named *buffer* (as they were at the time the @ command was executed) as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode buffer, the *ex* command parser shall behave as if the line was terminated by a <newline>.

If an error occurs during this process, or a line specified by the addresses does not exist when the current line would be set to it, or more than a single line was specified by the addresses, and the contents of the edit buffer are replaced (for example, by the *ex* :**edit** command) an error message shall be written, and no more commands resulting from the execution of this command shall be processed.

*Current line*: As specified for the individual *ex* commands.

*Current column*: As specified for the individual *ex* commands.

**Regular Expressions in ex**

The *ex* utility shall support regular expressions that are a superset of the basic regular expressions described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions. A null regular expression ("/") shall be equivalent to the last regular expression encountered.

Regular expressions can be used in addresses to specify lines and, in some commands (for example, the **substitute** command), to specify portions of a line to be substituted.

The following constructs can be used to enhance the basic regular expressions:

\< Match the beginning of a *word*. (See the definition of *word* at the beginning of **Command Descriptions in ex** (on page 368).)

\> Match the end of a *word*.

~ Match the replacement part of the last **substitute** command. The tilde ('~') character can be escaped in a regular expression to become a normal character with no special meaning. The backslash shall be discarded.

When the editor option **magic** is not set, the only characters with special meanings shall be '^' at the beginning of a pattern, '\$' at the end of a pattern, and '\'. The characters '.', '\*', '[', and '~' shall be treated as ordinary characters unless preceded by a '\'; when preceded by a '\' they shall regain their special meaning, or in the case of backslash, be handled as a single backslash. Backslashes used to escape other characters shall be discarded.

**Replacement Strings in ex**

The character '&' ('\&' if the editor option **magic** is not set) in the replacement string shall stand for the text matched by the pattern to be replaced. The character '~' ('\~' if **magic** is not set) shall be replaced by the replacement part of the previous **substitute** command. The sequence '\n', where *n* is an integer, shall be replaced by the text matched by the pattern enclosed in the *n*th set of parentheses '\(' and '\)'.

The strings '\l', '\u', '\L', and '\U' can be used to modify the case of elements in the replacement string (using the '\&' or "\"digit) notation. The string '\l' ('\u') shall cause

15165 the character that follows to be converted to lowercase (uppercase). The string '`\L`' ('`\U`')  
 15166 shall cause all characters subsequent to it to be converted to lowercase (uppercase) as they are  
 15167 inserted by the substitution until the string '`\e`' or '`\E`', or the end of the replacement string,  
 15168 is encountered.

15169 Otherwise, any character following a backslash shall be treated as that literal character, and the  
 15170 escaping backslash shall be discarded.

15171 An example of case conversion with the **s** command is as follows:

```
15172 :p
15173 The cat sat on the mat.
15174 :s/\<.at\>/\u&/gp
15175 The Cat Sat on the Mat.
15176 :s/S\ (. *\) M/S\U\1\eM/p
15177 The Cat SAT ON THE Mat.
```

## 15178 Edit Options in ex

15179 The *ex* utility has a number of options that modify its behavior. These options have default  
 15180 settings, which can be changed using the **set** command.

15181 Options are Boolean unless otherwise specified.

### 15182 **autoindent, ai**

15183 [Default *unset*]

15184 If **autoindent** is set, each line in input mode shall be indented (using first as many `<tab>`s as  
 15185 possible, as determined by the editor option **tabstop**, and then using `<space>`s) to align with  
 15186 another line, as follows:

- 15187 1. If in open or visual mode and the text input is part of a line-oriented command (see the  
 15188 EXTENDED DESCRIPTION in *vi*), align to the first column.
- 15189 2. Otherwise, if in open or visual mode, indentation for each line shall be set as follows:
  - 15190 a. If a line was previously inserted as part of this command, it shall be set to the  
 15191 indentation of the last inserted line by default, or as otherwise specified for the  
 15192 `<control>-D` character in **Input Mode Commands in vi** (on page 1022).
  - 15193 b. Otherwise, it shall be set to the indentation of the previous current line, if any;  
 15194 otherwise, to the first column.
- 15195 3. For the *ex* **a**, **i**, and **c** commands, indentation for each line shall be set as follows:
  - 15196 a. If a line was previously inserted as part of this command, it shall be set to the  
 15197 indentation of the last inserted line by default, or as otherwise specified for the *eof*  
 15198 character in **Scroll** (on page 366).
  - 15199 b. Otherwise, if the command is the *ex* **a** command, it shall be set to the line appended  
 15200 after, if any; otherwise to the first column.
  - 15201 c. Otherwise, if the command is the *ex* **i** command, it shall be set to the line inserted  
 15202 before, if any; otherwise to the first column.
  - 15203 d. Otherwise, if the command is the *ex* **c** command, it shall be set to the indentation of  
 15204 the line replaced.

15205      **autoprint, ap**15206      [Default *set*]

15207      If **autoprint** is set, the current line shall be written after each **ex** command that modifies the  
 15208      contents of the current edit buffer, and after each **tag** command for which the tag search pattern  
 15209      was found or tag line number was valid, unless:

- 15210          1. The command was executed while in open or visual mode.
- 15211          2. The command was executed as part of a **global** or **v** command or **@** buffer execution.
- 15212          3. The command was the form of the **read** command that reads a file into the edit buffer.
- 15213          4. The command was the **append**, **change**, or **insert** command.
- 15214          5. The command was not terminated by a <newline>.
- 15215          6. The current line shall be written by a flag specified to the command; for example, **delete #**  
 15216          shall write the current line as specified for the flag modifier to the **delete** command, and  
 15217          not as specified by the **autoprint** edit option.

15218      **autowrite, aw**15219      [Default *unset*]

15220      If **autowrite** is set, and the edit buffer has been modified since it was last completely written to  
 15221      any file, the contents of the edit buffer shall be written as if the **ex write** command had been  
 15222      specified without arguments, before each command affected by the **autowrite** edit option is  
 15223      executed. Appending the character '!' to the command name of any of the **ex** commands  
 15224      except '!' shall prevent the write. If the write fails, it shall be an error and the command shall  
 15225      not be executed.

15226      **beautify, bf**15227 xSI      [Default *unset*]

15228      If **beautify** is set, all non-printable characters, other than <tab>s, <newline>s, and <form-feed>s,  
 15229      shall be discarded from text read in from files.

15230      **directory, dir**15231      [Default *implementation-defined*]

15232      The value of this option specifies the directory in which the editor buffer is to be placed. If this  
 15233      directory is not writable by the user, the editor shall quit.

15234      **edcompatible, ed**15235      [Default *unset*]

15236      Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled  
 15237      by repeating the suffixes.

|       |                                                                                                                                        |
|-------|----------------------------------------------------------------------------------------------------------------------------------------|
| 15238 | <b>errorbells, eb</b>                                                                                                                  |
| 15239 | [Default <i>unset</i> ]                                                                                                                |
| 15240 | If the editor is in <b>ex</b> mode, and the terminal does not support a standout mode (such as inverse                                 |
| 15241 | video), and <b>errorbells</b> is set, error messages shall be preceded by alerting the terminal.                                       |
| 15242 | <b>exrc</b>                                                                                                                            |
| 15243 | [Default <i>unset</i> ]                                                                                                                |
| 15244 | If <b>exrc</b> is set, <i>ex</i> shall access any <b>.exrc</b> file in the current directory, as described in <b>Initialization in</b> |
| 15245 | <b>ex and vi</b> (on page 358). If <b>exrc</b> is not set, <i>ex</i> shall ignore any <b>.exrc</b> file in the current directory       |
| 15246 | during initialization, unless the current directory is that named by the <i>HOME</i> environment                                       |
| 15247 | variable.                                                                                                                              |
| 15248 | <b>ignorecase, ic</b>                                                                                                                  |
| 15249 | [Default <i>unset</i> ]                                                                                                                |
| 15250 | If <b>ignorecase</b> is set, characters that have uppercase and lowercase representations shall have                                   |
| 15251 | those representations considered as equivalent for purposes of regular expression comparison.                                          |
| 15252 | The <b>ignorecase</b> edit option shall affect all remembered regular expressions; for example,                                        |
| 15253 | unsetting the <b>ignorecase</b> edit option shall cause a subsequent <b>vi n</b> command to search for the                             |
| 15254 | last basic regular expression in a case-sensitive fashion.                                                                             |
| 15255 | <b>list</b>                                                                                                                            |
| 15256 | [Default <i>unset</i> ]                                                                                                                |
| 15257 | If <b>list</b> is set, edit buffer lines written while in <i>ex</i> command mode shall be written as specified for                     |
| 15258 | the <b>print</b> command with the <b>l</b> flag specified. In open or visual mode, each edit buffer line shall                         |
| 15259 | be displayed as specified for the <i>ex</i> <b>print</b> command with the <b>l</b> flag specified. In open or visual                   |
| 15260 | text input mode, when the cursor does not rest on any character in the line, it shall rest on the                                      |
| 15261 | ' <b>\$</b> ' marking the end of the line.                                                                                             |
| 15262 | <b>magic</b>                                                                                                                           |
| 15263 | [Default <i>set</i> ]                                                                                                                  |
| 15264 | If <b>magic</b> is set, modify the interpretation of characters in regular expressions and substitution                                |
| 15265 | replacement strings (see <b>Regular Expressions in ex</b> (on page 391) and <b>Replacement Strings in</b>                              |
| 15266 | <b>ex</b> (on page 391)).                                                                                                              |
| 15267 | <b>mesg</b>                                                                                                                            |
| 15268 | [Default <i>set</i> ]                                                                                                                  |
| 15269 | If <b>mesg</b> is set, the permission for others to use the <b>write</b> or <b>talk</b> commands to write to the                       |
| 15270 | terminal shall be turned on while in open or visual mode. The shell-level command <i>mesg n</i> shall                                  |
| 15271 | take precedence over any setting of the <i>ex</i> <b>mesg</b> option; that is, if <b>mesg y</b> was issued before the                  |
| 15272 | editor started (or in a shell escape), such as:                                                                                        |
| 15273 | :! <b>mesg y</b>                                                                                                                       |
| 15274 | the <b>mesg</b> option in <i>ex</i> shall suppress incoming messages, but the <b>mesg</b> option shall not enable                      |
| 15275 | incoming messages if <b>mesg n</b> was issued.                                                                                         |

## 15276 **number, nu**

15277 [Default *unset*]

15278 If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line  
 15279 numbers, in the format specified by the **print** command with the **#** flag specified. In *ex* text input  
 15280 mode, each line shall be preceded by the line number it will have in the file.

15281 In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in  
 15282 the format specified by the *ex* **print** command with the **#** flag specified. This line number shall  
 15283 not be considered part of the line for the purposes of evaluating the current column; that is,  
 15284 column position 1 shall be the first column position after the format specified by the **print**  
 15285 command.

## 15286 **paragraphs, para**

15287 [Default in the POSIX locale `IPLPPPQPP LIpplpipbp`]

15288 The **paragraphs** edit option shall define additional paragraph boundaries for the open and visual  
 15289 mode commands. The **paragraphs** edit option can be set to a character string consisting of zero  
 15290 or more character pairs. It shall be an error to set it to an odd number of characters.

## 15291 **prompt**

15292 [Default *set*]

15293 If **prompt** is set, *ex* command mode input shall be prompted for with a colon (`:`); when unset,  
 15294 no prompt shall be written.

## 15295 **readonly**

15296 [Default *see text*]

15297 If the **readonly** edit option is set, read-only mode shall be enabled (see **Write** (on page 386)). The  
 15298 **readonly** edit option shall be initialized to set if either of the following conditions are true:

- 15299 • The command-line option `-R` was specified.
- 15300 • Performing actions equivalent to the `access()` function called with the following arguments  
 15301 indicates that the file lacks write permission:
  - 15302 1. The current pathname is used as the *path* argument.
  - 15303 2. The constant **W\_OK** is used as the *amode* argument.

15304 The **readonly** edit option may be initialized to set for other, implementation-defined reasons.  
 15305 The **readonly** edit option shall not be initialized to unset based on any special privileges of the  
 15306 user or process. The **readonly** edit option shall be reinitialized each time that the contents of the  
 15307 edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly  
 15308 set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again  
 15309 be reinitialized each time that the contents of the edit buffer are replaced.

|       |                                                                                                                                                         |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| 15310 | <b>redraw</b>                                                                                                                                           |
| 15311 | [Default <i>unset</i> ]                                                                                                                                 |
| 15312 | The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a                                                     |
| 15313 | large amount of output to the terminal, it is useful only at high transmission speeds.)                                                                 |
| 15314 | <b>remap</b>                                                                                                                                            |
| 15315 | [Default <i>set</i> ]                                                                                                                                   |
| 15316 | If <b>remap</b> is set, map translation shall allow for maps defined in terms of other maps; translation                                                |
| 15317 | shall continue until a final product is obtained. If unset, only a one-step translation shall be done.                                                  |
| 15318 | <b>report</b>                                                                                                                                           |
| 15319 | [Default 5]                                                                                                                                             |
| 15320 | The value of this <b>report</b> edit option specifies what number of lines being added, copied, deleted,                                                |
| 15321 | or modified in the edit buffer will cause an informational message to be written to the user. The                                                       |
| 15322 | following conditions shall cause an informational message. The message shall contain the                                                                |
| 15323 | number of lines added, copied, deleted, or modified, but is otherwise unspecified.                                                                      |
| 15324 | • An <i>ex</i> or <i>vi</i> editor command, other than <b>open</b> , <b>undo</b> , or <b>visual</b> , that modifies at least the value                  |
| 15325 | of the <b>report</b> edit option number of lines, and which is not part of an <i>ex</i> <b>global</b> or <b>v</b>                                       |
| 15326 | command, or <i>ex</i> or <i>vi</i> buffer execution, shall cause an informational message to be written.                                                |
| 15327 | • An <i>ex</i> <b>yank</b> or <i>vi</i> <b>y</b> or <b>Y</b> command, that copies at least the value of the <b>report</b> edit option plus              |
| 15328 | 1 number of lines, and which is not part of an <i>ex</i> <b>global</b> or <b>v</b> command, or <i>ex</i> or <i>vi</i> buffer                            |
| 15329 | execution, shall cause an informational message to be written.                                                                                          |
| 15330 | • An <i>ex</i> <b>global</b> , <b>v</b> , <b>open</b> , <b>undo</b> , or <b>visual</b> command or <i>ex</i> or <i>vi</i> buffer execution, that adds or |
| 15331 | deletes a total of at least the value of the <b>report</b> edit option number of lines, and which is not                                                |
| 15332 | part of an <i>ex</i> <b>global</b> or <b>v</b> command, or <i>ex</i> or <i>vi</i> buffer execution, shall cause an informational                        |
| 15333 | message to be written. (For example, if 3 lines were added and 8 lines deleted during an <i>ex</i>                                                      |
| 15334 | <b>visual</b> command, 5 would be the number compared against the <b>report</b> edit option after the                                                   |
| 15335 | command completed.)                                                                                                                                     |
| 15336 | <b>scroll, scr</b>                                                                                                                                      |
| 15337 | [Default (number of lines in the display −1)/2]                                                                                                         |
| 15338 | The value of the <b>scroll</b> edit option shall determine the number of lines scrolled by the <i>ex</i>                                                |
| 15339 | <control>-D and <b>z</b> commands. For the <i>vi</i> <control>-D and <control>-U commands, it shall be the                                              |
| 15340 | initial number of lines to scroll when no previous <control>-D or <control>-U command has                                                               |
| 15341 | been executed.                                                                                                                                          |
| 15342 | <b>sections</b>                                                                                                                                         |
| 15343 | [Default in the POSIX locale <code>NHSHH HUnhsh</code> ]                                                                                                |
| 15344 | The <b>sections</b> edit option shall define additional section boundaries for the open and visual mode                                                 |
| 15345 | commands. The <b>sections</b> edit option can be set to a character string consisting of zero or more                                                   |
| 15346 | character pairs; it shall be an error to set it to an odd number of characters.                                                                         |



**15347 shell, sh**

15348 [Default from the environment variable *SHELL*]

15349 The value of this option shall be a string. The default shall be taken from the *SHELL*  
15350 environment variable. If the *SHELL* environment variable is null or empty, the *sh* (see *sh*) utility  
15351 shall be the default.

**15352 shiftwidth, sw**

15353 [Default 8]

15354 The value of this option shall give the width in columns of an indentation level used during  
15355 autoindentation and by the shift commands (< and >).

**15356 showmatch, sm**

15357 [Default *unset*]

15358 The functionality described for the **showmatch** edit option need not be supported on block-  
15359 mode terminals or terminals with insufficient capabilities.

15360 If **showmatch** is set, in open or visual mode, when a ' ' or '}' is typed, if the matching ' ( ' or  
15361 ' { ' is currently visible on the display, the matching ' ( ' or ' { ' shall be flagged moving the  
15362 cursor to its location for an unspecified amount of time.

**15363 showmode**

15364 [Default *unset*]

15365 If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be  
15366 displayed on the last line of the display. Command mode and text input mode shall be  
15367 differentiated; other unspecified modes and implementation-defined information may be  
15368 displayed.

**15369 slowopen**

15370 [Default *unset*]

15371 If **slowopen** is set during open and visual text input modes, the editor shall not update portions  
15372 of the display other than those display line columns that display the characters entered by the  
15373 user (see **Input Mode Commands in vi** (on page 1022)).

**15374 tabstop, ts**

15375 [Default 8]

15376 The value of this edit option shall specify the column boundary used by a <tab> in the display  
15377 (see **autoprint, ap** (on page 393) and **Input Mode Commands in vi** (on page 1022)).

**15378 taglength, tl**

15379 [Default zero]

15380 The value of this edit option shall specify the maximum number of characters that are  
15381 considered significant in the user-specified tag name and in the tag name from the tags file. If the  
15382 value is zero, all characters in both tag names shall be significant.

15383       **tags**  
15384       [Default *see text*]  
15385       The value of this edit option shall be a string of <blank>-delimited pathnames of files used by  
15386       the **tag** command. The default value is unspecified.

15387       **term**  
15388       [Default from the environment variable *TERM*]  
15389       The value of this edit option shall be a string. The default shall be taken from the *TERM* variable  
15390       in the environment. If the *TERM* environment variable is empty or null, the default is  
15391       unspecified. The editor shall use the value of this edit option to determine the type of the display  
15392       device.  
15393       The results are unspecified if the user changes the value of the term edit option after editor  
15394       initialization.

15395       **terse**  
15396       [Default *unset*]  
15397       If **terse** is set, error messages may be less verbose. However, except for this caveat, error  
15398       messages are unspecified. Furthermore, not all error messages need change for different settings  
15399       of this option.

15400       **warn**  
15401       [Default *set*]  
15402       If **warn** is set, and the contents of the edit buffer have been modified since they were last  
15403       completely written, the editor shall write a warning message before certain ! commands (see  
15404       **Escape** (on page 389)).

15405       **window**  
15406       [Default *see text*]  
15407       A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in  
15408       visual mode, to specify the number of lines displayed when the screen is repainted.  
15409       If the **-w** command-line option is not specified, the default value shall be set to the value of the  
15410       *LINES* environment variable. If the *LINES* environment variable is empty or null, the default  
15411       shall be the number of lines in the display minus 1.  
15412       Setting the **window** edit option to zero or to a value greater than the number of lines in the  
15413       display minus 1 (either explicitly or based on the **-w** option or the *LINES* environment variable)  
15414       shall cause the **window** edit option to be set to the number of lines in the display minus 1.  
15415       The baud rate of the terminal line may change the default in an implementation-defined manner.

## 15416 **wrapmargin, wm**

15417 [Default 0]

15418 If the value of this edit option is zero, it shall have no effect.

15419 If not in the POSIX locale, the effect of this edit option is implementation-defined.

15420 Otherwise, it shall specify a number of columns from the ending margin of the terminal.

15421 During open and visual text input modes, for each character for which any part of the character  
15422 is displayed in a column that is less than **wrapmargin** columns from the ending margin of the  
15423 display line, the editor shall behave as follows:

15424 1. If the character triggering this event is a <blank>, it, and all immediately preceding  
15425 <blank>s on the current line entered during the execution of the current text input  
15426 command, shall be discarded, and the editor shall behave as if the user had entered a single  
15427 <newline> instead. In addition, if the next user-entered character is a <space>, it shall be  
15428 discarded as well.

15429 2. Otherwise, if there are one or more <blank>s on the current line immediately preceding the  
15430 last group of inserted non-<blank>s which was entered during the execution of the current  
15431 text input command, the <blank>s shall be replaced as if the user had entered a single  
15432 <newline> instead.

15433 If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any  
15434 <blank>s at or after the cursor in the current line shall be discarded.

15435 The ending margin shall be determined by the system or overridden by the user, as described for  
15436 **COLUMNS** in the ENVIRONMENT VARIABLES section and the Base Definitions volume of  
15437 IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

## 15438 **wrapscan, ws**

15439 [Default set]

15440 If **wrapscan** is set, searches (the *ex* / or ? addresses, or open and visual mode /, ?, **N**, and **n**  
15441 commands) shall wrap around the beginning or end of the edit buffer; when unset, searches  
15442 shall stop at the beginning or end of the edit buffer.

## 15443 **writeany, wa**

15444 [Default unset]

15445 If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be  
15446 inhibited, as described in editor option **autowrite**.

## 15447 **EXIT STATUS**

15448 The following exit values shall be returned:

15449 0 Successful completion.

15450 >0 An error occurred.

## 15451 **CONSEQUENCES OF ERRORS**

15452 When any error is encountered and the standard input is not a terminal device file, *ex* shall not  
15453 write the file or return to command or text input mode, and shall terminate with a non-zero exit  
15454 status.

15455 Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP  
15456 asynchronous event.

15457 Otherwise, when an error is encountered, the editor shall behave as specified in **Command Line**  
 15458 **Parsing in ex** (on page 362).

#### 15459 APPLICATION USAGE

15460 If a SIGSEGV signal is received while *ex* is saving a file, the file might not be successfully saved.

15461 The **next** command can accept more than one file, so usage such as:

15462 `next `ls [abc]*``

15463 is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect  
 15464 only one file and unspecified results occur.

#### 15465 EXAMPLES

15466 None.

#### 15467 RATIONALE

15468 The *ex/vi* specification is based on the historical practice found in the 4 BSD and System V  
 15469 implementations of *ex* and *vi*. A freely redistributable implementation of *ex/vi*, which is  
 15470 tracking IEEE Std 1003.1-2001 fairly closely, and demonstrates the intended changes between  
 15471 historical implementations and IEEE Std 1003.1-2001, may be obtained by anonymous FTP from:

15472 `ftp://ftp.rdg.opengroup.org/pub/mirrors/nvi`

15473 A *restricted editor* (both the historical *red* utility and modifications to *ex*) were considered and  
 15474 rejected for inclusion. Neither option provided the level of security that users might expect.

15475 It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to  
 15476 implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor  
 15477 addressing; thus, it is not a mandatory requirement that such features should work on all  
 15478 terminals. It is the intention, however, that an *ex* implementation should provide the full set of  
 15479 capabilities on all terminals capable of supporting them.

#### 15480 Options

15481 The **-c** replacement for **+command** was inspired by the **-e** option of *sed*. Historically, all such  
 15482 commands (see **edit** and **next** as well) were executed from the last line of the edit buffer. This  
 15483 meant, for example, that **"/pattern"** would fail unless the **wrapsan** option was set.  
 15484 IEEE Std 1003.1-2001 requires conformance to historical practice. Historically, some  
 15485 implementations restricted the *ex* commands that could be listed as part of the command line  
 15486 arguments. For consistency, IEEE Std 1003.1-2001 does not permit these restrictions.

15487 In historical implementations of the editor, the **-R** option (and the **readonly** edit option) only  
 15488 prevented overwriting of files; appending to files was still permitted, mapping loosely into the  
 15489 *csh* **noclobber** variable. Some implementations, however, have not followed this semantic, and  
 15490 **readonly** does not permit appending either. IEEE Std 1003.1-2001 follows the latter practice,  
 15491 believing that it is a more obvious and intuitive meaning of **readonly**.

15492 The **-s** option suppresses all interactive user feedback and is useful for editing scripts in batch  
 15493 jobs. The list of specific effects is historical practice. The terminal type "incapable of supporting  
 15494 open and visual modes" has historically been named "dumb".

15495 The **-t** option was required because the *ctags* utility appears in IEEE Std 1003.1-2001 and the  
 15496 option is available in all historical implementations of *ex*.

15497 Historically, the *ex* and *vi* utilities accepted a **-x** option, which did encryption based on the  
 15498 algorithm found in the historical *crypt* utility. The **-x** option for encryption, and the associated  
 15499 *crypt* utility, were omitted because the algorithm used was not specifiable and the export control  
 15500 laws of some nations make it difficult to export cryptographic technology. In addition, it did not

15501 historically provide the level of security that users might expect.

### 15502 **Standard Input**

15503 An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file  
15504 character, <control>-D, is historically an *ex* command.

15505 There was no maximum line length in historical implementations of *ex*. Specifically, as it was  
15506 parsed in chunks, the addresses had a different maximum length than the filenames. Further, the  
15507 maximum line buffer size was declared as BUFSIZ, which was different lengths on different  
15508 systems. This version selected the value of {LINE\_MAX} to impose a reasonable restriction on  
15509 portable usage of *ex* and to aid test suite writers in their development of realistic tests that  
15510 exercise this limit.

### 15511 **Input Files**

15512 It was an explicit decision by the standard developers that a <newline> be added to any file  
15513 lacking one. It was believed that this feature of *ex* and *vi* was relied on by users in order to make  
15514 text files lacking a trailing <newline> more portable. It is recognized that this will require a  
15515 user-specified option or extension for implementations that permit *ex* and *vi* to edit files of type  
15516 other than text if such files are not otherwise identified by the system. It was agreed that the  
15517 ability to edit files of arbitrary type can be useful, but it was not considered necessary to  
15518 mandate that an *ex* or *vi* implementation be required to handle files other than text files.

15519 The paragraph in the INPUT FILES section, “By default, ...”, is intended to close a long-standing  
15520 security problem in *ex* and *vi*; that of the “modeline” or “modelines” edit option. This feature  
15521 allows any line in the first or last five lines of the file containing the strings "*ex*:" or "*vi*:"  
15522 (and, apparently, "*ei*:" or "*vx*:") to be a line containing editor commands, and *ex* interprets all  
15523 the text up to the next ':' or <newline> as a command. Consider the consequences, for  
15524 example, of an unsuspecting user using *ex* or *vi* as the editor when replying to a mail message in  
15525 which a line such as:

15526 *ex*:! rm -rf :

15527 appeared in the signature lines. The standard developers believed strongly that an editor should  
15528 not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from  
15529 their implementations of *ex* and *vi*.

### 15530 **Asynchronous Events**

15531 The intention of the phrase “complete write” is that the entire edit buffer be written to stable  
15532 storage. The note regarding temporary files is intended for implementations that use temporary  
15533 files to back edit buffers unnamed by the user.

15534 Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the **Q** command in visual  
15535 mode; that is, it exited visual mode and entered *ex* mode. IEEE Std 1003.1-2001 permits, but does  
15536 not require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text  
15537 input mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi*  
15538 alerted the terminal on this event, and some did not. IEEE Std 1003.1-2001 requires that SIGINT  
15539 behave identically to <ESC>, and that the terminal not be alerted.

15540 Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as  
15541 completed lines were retained, but any partial line discarded, and the editor returned to  
15542 command mode. IEEE Std 1003.1-2001 is silent on this issue; implementations are encouraged to  
15543 follow historical practice, where possible.

Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore impossible to suspend the editor in visual text input mode. There are two major reasons for this. The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where the shell *execs* an application that then *execs vi* usually caused confusion for the terminal state if SIGTSTP was delivered to the process group in the default manner. The second was that most implementations of the UNIX *curses* package are not reentrant, and the receipt of SIGTSTP at the wrong time will cause them to crash. IEEE Std 1003.1-2001 is silent on this issue; implementations are encouraged to treat suspension as an asynchronous event if possible.

Historically, modifications to the edit buffer made before SIGINT interrupted an operation were retained; that is, anywhere from zero to all of the lines to be modified might have been modified by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT. IEEE Std 1003.1-2001 permits this behavior, noting that the **undo** command is required to be able to undo these partially completed commands.

The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is unspecified because some implementations attempt to save the edit buffer in a useful state when other signals are received.

### Standard Error

For *ex/vi*, diagnostic messages are those messages reported as a result of a failed attempt to invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination condition. Diagnostic messages should not be confused with the error messages generated by inappropriate or illegal user commands.

### Initialization in *ex* and *vi*

If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the alternate and current pathnames will be set. Informally, they are set as follows:

1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the current pathname will be set to the filename argument (the first filename argument in the case of the **next** command) and the alternate pathname will be set to the previous current pathname, if there was one.
2. In the case of the file read/write forms of the **read** and **write** commands, if there is no current pathname, the current pathname will be set to the filename argument.
3. Otherwise, the alternate pathname will be set to the filename argument.

For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there was a previous current pathname, the alternate pathname. The commands **:write**, **!command**, and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for some reason, the alternate pathname would be set. The **read** and **write** commands set the alternate pathname to their *file* argument, unless the current pathname is not set, in which case they set the current pathname to their *file* arguments. The alternate pathname was not historically set by the **:source** command. IEEE Std 1003.1-2001 requires conformance to historical practice. Implementations adding commands that take filenames as arguments are encouraged to set the alternate pathname as described here.

Historically, *ex* and *vi* read the **.exrc** file in the *\$HOME* directory twice, if the editor was executed in the *\$HOME* directory. IEEE Std 1003.1-2001 prohibits this behavior.

Historically, the 4 BSD *ex* and *vi* read the *\$HOME* and local **.exrc** files if they were owned by the real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was a security problem because it is possible to put normal UNIX system commands inside a **.exrc**

15589 file. IEEE Std 1003.1-2001 does not specify the **sourceany** option, and historical implementations  
15590 are encouraged to delete it.

15591 The **.exrc** files must be owned by the real ID of the user, and not writable by anyone other than  
15592 the owner. The appropriate privileges exception is intended to permit users to acquire special  
15593 privileges, but continue to use the **.exrc** files in their home directories.

15594 System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is  
15595 that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off,  
15596 so by default, local **.exrc** files were not read. The problem this was intended to solve was that  
15597 System V permitted users to give away files, so there is no possible ownership or writeability  
15598 test to ensure that the file is safe. This is still a security problem on systems where users can give  
15599 away files, but there is nothing additional that IEEE Std 1003.1-2001 can do. The  
15600 implementation-defined exception is intended to permit groups to have local **.exrc** files that are  
15601 shared by users, by creating pseudo-users to own the shared files.

15602 IEEE Std 1003.1-2001 does not mention system-wide *ex* and *vi* start-up files. While they exist in  
15603 several implementations of *ex* and *vi*, they are not present in any implementations considered  
15604 historical practice by IEEE Std 1003.1-2001. Implementations that have such files should use  
15605 them only if they are owned by the real user ID or an appropriate user (for example, root on  
15606 UNIX systems) and if they are not writable by any user other than their owner. System-wide  
15607 start-up files should be read before the *EXINIT* variable, **\$HOME/.exrc**, or local **.exrc** files are  
15608 evaluated.

15609 Historically, any *ex* command could be entered in the *EXINIT* variable or the **.exrc** file, although  
15610 ones requiring that the edit buffer already contain lines of text generally caused historical  
15611 implementations of the editor to drop **core**. IEEE Std 1003.1-2001 requires that any *ex* command  
15612 be permitted in the *EXINIT* variable and **.exrc** files, for simplicity of specification and  
15613 consistency, although many of them will obviously fail under many circumstances.

15614 The initialization of the contents of the edit buffer uses the phrase “the effect shall be” with  
15615 regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded  
15616 during the initialization phase not be lost; that is, loading the edit buffer should fail if the **.exrc**  
15617 file read in the contents of a file and did not subsequently write the edit buffer. An additional  
15618 intent of this phrase is to specify that the initial current line and column is set as specified for the  
15619 individual *ex* commands.

15620 Historically, the **-t** option behaved as if the tag search were a **+command**; that is, it was executed  
15621 from the last line of the file specified by the tag. This resulted in the search failing if the pattern  
15622 was a forward search pattern and the **wrapscan** edit option was not set. IEEE Std 1003.1-2001  
15623 does not permit this behavior, requiring that the search for the tag pattern be performed on the  
15624 entire file, and, if not found, that the current line be set to a more reasonable location in the file.

15625 Historically, the empty edit buffer presented for editing when a file was not specified by the user  
15626 was unnamed. This is permitted by IEEE Std 1003.1-2001; however, implementations are  
15627 encouraged to provide users a temporary filename for this buffer because it permits them the  
15628 use of *ex* commands that use the current pathname during temporary edit sessions.

15629 Historically, the file specified using the **-t** option was not part of the current argument list. This  
15630 practice is permitted by IEEE Std 1003.1-2001; however, implementations are encouraged to  
15631 include its name in the current argument list for consistency.

15632 Historically, the **-c** command was generally not executed until a file that already exists was  
15633 edited. IEEE Std 1003.1-2001 requires conformance to this historical practice. Commands that  
15634 could cause the **-c** command to be executed include the *ex* commands **edit**, **next**, **recover**,  
15635 **rewind**, and **tag**, and the *vi* commands **<control>-^** and **<control>-J**. Historically, reading a file  
15636 into an edit buffer did not cause the **-c** command to be executed (even though it might set the

current pathname) with the exception that it did cause the `-c` command to be executed if: the editor was in *ex* mode, the edit buffer had no current pathname, the edit buffer was empty, and no read commands had yet been attempted. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

Historically, the `-r` option was the same as a normal edit session if there was no recovery information available for the file. This allowed users to enter:

```
vi -r *.c
```

and recover whatever files were recoverable. In some implementations, recovery was attempted only on the first file named, and the file was not entered into the argument list; in others, recovery was attempted for each file named. In addition, some historical implementations ignored `-r` if `-t` was specified or did not support command line *file* arguments with the `-t` option. For consistency and simplicity of specification, IEEE Std 1003.1-2001 disallows these special cases, and requires that recovery be attempted the first time each file is edited.

Historically, *vi* initialized the ‘ and ’ marks, but *ex* did not. This meant that if the first command in *ex* mode was **visual** or if an *ex* command was executed first (for example, *vi* +10 *file*), *vi* was entered without the marks being initialized. Because the standard developers believed the marks to be generally useful, and for consistency and simplicity of specification, IEEE Std 1003.1-2001 requires that they always be initialized if in open or visual mode, or if in *ex* mode and the edit buffer is not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice; however, it has always been possible to set (and use) marks in empty edit buffers in open and visual mode edit sessions.

## Addressing

Historically, *ex* and *vi* accepted the additional addressing forms ‘\/' and ‘\?’. They were equivalent to “//” and “??”, respectively. They are not required by IEEE Std 1003.1-2001, mostly because nobody can remember whether they ever did anything different historically.

Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the % address in empty files for others. For consistency, IEEE Std 1003.1-2001 requires support for the former in the few commands where it makes sense, and disallows it otherwise. In addition, because IEEE Std 1003.1-2001 requires that % be logically equivalent to “1,\$”, it is also supported where it makes sense and disallowed otherwise.

Historically, the % address could not be followed by further addresses. For consistency and simplicity of specification, IEEE Std 1003.1-2001 requires that additional addresses be supported.

All of the following are valid *addresses*:

```
+++ Three lines after the current line.
/re/- One line before the next occurrence of re.
-2 Two lines before the current line.
3 ---- 2 Line one (note intermediate negative address).
1 2 3 Line six.
```

Any number of addresses can be provided to commands taking addresses; for example, “1,2,3,4,5p” prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the **print** command. This, in combination with the semicolon delimiter, permits users to create commands based on ordered patterns in the file. For example, the command **3/foo/+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next two lines. Note that the address **3**; must be evaluated before being discarded because the search



origin for the **/foo/** command depends on this.

Historically, values could be added to addresses by including them after one or more <blank>s; for example, **3 – 5p** wrote the seventh line of the file, and **/foo/ 5** was the same as **/foo/+5**. However, only absolute values could be added; for example, **5 /foo/** was an error. IEEE Std 1003.1-2001 requires conformance to historical practice. Address offsets are separately specified from addresses because they could historically be provided to visual mode search commands.

Historically, any missing addresses defaulted to the current line. This was true for leading and trailing comma-delimited addresses, and for trailing semicolon-delimited addresses. For consistency, IEEE Std 1003.1-2001 requires it for leading semicolon addresses as well.

Historically, *ex* and *vi* accepted the '^' character as both an address and as a flag offset for commands. In both cases it was identical to the '-' character. IEEE Std 1003.1-2001 does not require or prohibit this behavior.

Historically, the enhancements to basic regular expressions could be used in addressing; for example, '~', '\<', and '\>'. IEEE Std 1003.1-2001 requires conformance to historical practice; that is, that regular expression usage be consistent, and that regular expression enhancements be supported wherever regular expressions are used.

### Command Line Parsing in ex

Historical *ex* command parsing was even more complex than that described here. IEEE Std 1003.1-2001 requires the subset of the command parsing that the standard developers believed was documented and that users could reasonably be expected to use in a portable fashion, and that was historically consistent between implementations. (The discarded functionality is obscure, at best.) Historical implementations will require changes in order to comply with IEEE Std 1003.1-2001; however, users are not expected to notice any of these changes. Most of the complexity in *ex* parsing is to handle three special termination cases:

1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by <newline>s (they can contain vertical-line characters that are usually shell pipes).
2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands, optionally containing vertical-line characters, as their first arguments.
3. The **s** command takes a regular expression as its first argument, and uses the delimiting characters to delimit the command.

Historically, vertical-line characters in the *+command* argument of the **ex**, **edit**, **next**, **vi**, and **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did not delimit the command at all. For example, the following commands are all valid:

```
:edit +25 | s/abc/ABC/ file.c
:s/ | /PIPE/
:read !spell % | columnate
:global/pattern/p | l
:s/a/b/ | s/c/d | set
```

Historically, empty or <blank> filled lines in **.exrc** files and **sourced** files (as well as *EXINIT* variables and *ex* command scripts) were treated as default commands; that is, **print** commands. IEEE Std 1003.1-2001 specifically requires that they be ignored when encountered in **.exrc** and **sourced** files to eliminate a common source of new user error.

Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were handled oddly when executed from *ex* mode. For example, the command `||| <carriage-return>`, when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command `|` would only display the line after the next line, instead of the next two lines. The former worked more logically when executed from *vi* mode, and displayed lines 2, 3, and 4. IEEE Std 1003.1-2001 requires the *vi* behavior; that is, a single default command and line number increment for each command separator, and trailing <newline>s after vertical-line separators are discarded.

Historically, *ex* permitted a single extra colon as a leading command character; for example, `:g/pattern/:p` was a valid command. IEEE Std 1003.1-2001 generalizes this to require that any number of leading colon characters be stripped.

Historically, any prefix of the **delete** command could be followed without intervening <blank>s by a flag character because in the command `d p`, *p* is interpreted as the buffer *p*. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, the **k** command could be followed by the mark name without intervening <blank>s. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, the **s** command could be immediately followed by flag and option characters; for example, `s/e/E/|s|sgc3p` was a valid command. However, flag characters could not stand alone; for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would succeed. (Obviously, the '#' flag character was used as a delimiter character if it followed the command.) Another issue was that option characters had to precede flag characters even when the command was fully specified; for example, the command `s/e/E/pg` would fail, while the command `s/e/E/gp` would succeed. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, the first command name that had a prefix matching the input from the user was the executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command. Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**. IEEE Std 1003.1-2001 requires conformance to historical practice. The restriction on command search order for implementations with extensions is to avoid the addition of commands such that the historical prefixes would fail to work portably.

Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands, separated by vertical-line characters, that entered or exited visual mode or the editor. Because implementations of *vi* exist that do not exhibit this failure mode, IEEE Std 1003.1-2001 does not permit it.

The requirement that alphabetic command names consist of all following alphabetic characters up to the next non-alphabetic character means that alphabetic command names must be separated from their arguments by one or more non-alphabetic characters, normally a <blank> or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*, <newline>, <carriage-return>) erased any prompting character and displayed the next lines without scrolling the terminal; that is, immediately below any previously displayed lines. This provided a cleaner presentation of the lines in the file for the user. IEEE Std 1003.1-2001 does not require this behavior because it may be impossible in some situations; however, implementations are strongly encouraged to provide this semantic if possible.

Historically, it was possible to change files in the middle of a command, and have the rest of the command executed in the new file; for example:

```
15771 :edit +25 file.c | s/abc/ABC/ | 1
```

15772 was a valid command, and the substitution was attempted in the newly edited file.  
 15773 IEEE Std 1003.1-2001 requires conformance to historical practice. The following commands are  
 15774 examples that exercise the **ex** parser:

```
15775 echo 'foo | bar' > file1; echo 'foo/bar' > file2;
```

```
15776 vi
```

```
15777 :edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\/SLASH/ | wq
```

15778 Historically, there was no protection in editor implementations to avoid **ex** **global**, **v**, **@**, or **\***  
 15779 commands changing edit buffers during execution of their associated commands. Because this  
 15780 would almost invariably result in catastrophic failure of the editor, and implementations exist  
 15781 that do exhibit these problems, IEEE Std 1003.1-2001 requires that changing the edit buffer  
 15782 during a **global** or **v** command, or during a **@** or **\*** command for which there will be more than a  
 15783 single execution, be an error. Implementations supporting multiple edit buffers simultaneously  
 15784 are strongly encouraged to apply the same semantics to switching between buffers as well.

15785 The **ex** command quoting required by IEEE Std 1003.1-2001 is a superset of the quoting in  
 15786 historical implementations of the editor. For example, it was not historically possible to escape a  
 15787 <blank> in a filename; for example, **:edit foo\\\ bar** would report that too many filenames had  
 15788 been entered for the edit command, and there was no method of escaping a <blank> in the first  
 15789 argument of an **edit**, **ex**, **next**, or **visual** command at all. IEEE Std 1003.1-2001 extends historical  
 15790 practice, requiring that quoting behavior be made consistent across all **ex** commands, except for  
 15791 the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V  
 15792 instead of backslashes for quoting. For those four commands, IEEE Std 1003.1-2001 requires  
 15793 conformance to historical practice.

15794 Backslash quoting in **ex** is non-intuitive. Backslash escapes are ignored unless they escape a  
 15795 special character; for example, when performing *file* argument expansion, the string "\\%" is  
 15796 equivalent to '\%', not "\<current pathname>". This can be confusing for users because  
 15797 backslash is usually one of the characters that causes shell expansion to be performed, and  
 15798 therefore shell quoting rules must be taken into consideration. Generally, quoting characters are  
 15799 only considered if they escape a special character, and a quoting character must be provided for  
 15800 each layer of parsing for which the character is special. As another example, only a single  
 15801 backslash is necessary for the '\1' sequence in substitute replacement patterns, because the  
 15802 character '1' is not special to any parsing layer above it.

15803 <control>-V quoting in **ex** is slightly different from backslash quoting. In the four commands  
 15804 where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character  
 15805 may be escaped by a <control>-V whether it would have a special meaning or not.  
 15806 IEEE Std 1003.1-2001 requires conformance to historical practice.

15807 Historical implementations of the editor did not require delimiters within character classes to be  
 15808 escaped; for example, the command **:s/[//** on the string "xxx/yyy" would delete the '/' from  
 15809 the string. IEEE Std 1003.1-2001 disallows this historical practice for consistency and because it  
 15810 places a large burden on implementations by requiring that knowledge of regular expressions be  
 15811 built into the editor parser.

15812 Historically, quoting <newline>s in **ex** commands was handled inconsistently. In most cases, the  
 15813 <newline> always terminated the command, regardless of any preceding escape character,  
 15814 because backslash characters did not escape <newline>s for most **ex** commands. However, some  
 15815 **ex** commands (for example, **s**, **map**, and **abbreviation**) permitted <newline>s to be escaped  
 15816 (although in the case of **map** and **abbreviation**, <control>-V characters escaped them instead of  
 15817 backslashes). This was true in not only the command line, but also **.exrc** and **sourced** files. For  
 15818 example, the command:

15819 map = foo<control-V><newline>bar

15820 would succeed, although it was sometimes difficult to get the <control>-V and the inserted  
 15821 <newline> passed to the *ex* parser. For consistency and simplicity of specification,  
 15822 IEEE Std 1003.1-2001 requires that it be possible to escape <newline>s in *ex* commands at all  
 15823 times, using backslashes for most *ex* commands, and using <control>-V characters for the **map**  
 15824 and **abbreviation** commands. For example, the command **print**<newline>**list** is required to be  
 15825 parsed as the single command **print**<newline>**list**. While this differs from historical practice,  
 15826 IEEE Std 1003.1-2001 developers believed it unlikely that any script or user depended on the  
 15827 historical behavior.

15828 Historically, an error in a command specified using the **-c** option did not cause the rest of the **-c**  
 15829 commands to be discarded. IEEE Std 1003.1-2001 disallows this for consistency with mapped  
 15830 keys, the **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc**  
 15831 files.

### 15832 **Input Editing in *ex***

15833 One of the common uses of the historical *ex* editor is over slow network connections. Editors  
 15834 that run in canonical mode can require far less traffic to and from, and far less processing on, the  
 15835 host machine, as well as more easily supporting block-mode terminals. For these reasons,  
 15836 IEEE Std 1003.1-2001 requires that *ex* be implemented using canonical mode input processing, as  
 15837 was done historically.

15838 IEEE Std 1003.1-2001 does not require the historical 4 BSD input editing characters “word erase”  
 15839 or “literal next”. For this reason, it is unspecified how they are handled by *ex*, although they  
 15840 must have the required effect. Implementations that resolve them after the line has been ended  
 15841 using a <newline> or <control>-M character, and implementations that rely on the underlying  
 15842 system terminal support for this processing, are both conforming. Implementations are strongly  
 15843 urged to use the underlying system functionality, if at all possible, for compatibility with other  
 15844 system text input interfaces.

15845 Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor  
 15846 moved to display the new end of the **autoindent** characters, but did not move the cursor to a  
 15847 new line, nor did it erase the <control>-D character from the line. IEEE Std 1003.1-2001 does not  
 15848 specify that the cursor remain on the same line or that the rest of the line is erased; however,  
 15849 implementations are strongly encouraged to provide the best possible user interface; that is, the  
 15850 cursor should remain on the same line, and any <control>-D character on the line should be  
 15851 erased.

15852 IEEE Std 1003.1-2001 does not require the historical 4 BSD input editing character “reprint”,  
 15853 traditionally <control>-R, which redisplayed the current input from the user. For this reason,  
 15854 and because the functionality cannot be implemented after the line has been terminated by the  
 15855 user, IEEE Std 1003.1-2001 makes no requirements about this functionality. Implementations are  
 15856 strongly urged to make this historical functionality available, if possible.

15857 Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*.  
 15858 IEEE Std 1003.1-2001 requires conformance to historical practice to avoid breaking historical *ex*  
 15859 scripts and **.exrc** files.

**eof**

Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left unspecified so that implementations can conform in the presence of systems that do not support this functionality. Implementations are encouraged to modify the line and redisplay it immediately, if possible.

The specification of the handling of the *eof* character differs from historical practice only in that *eof* characters are not discarded if they follow normal characters in the text input. Historically, they were always discarded.

**Command Descriptions in ex**

Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and **↵**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or command execution in an empty file, make sense only for commands that add new text to the edit buffer or write commands (because users may wish to write empty files). IEEE Std 1003.1-2001 requires this behavior for such commands and disallows it otherwise, for consistency and simplicity of specification.

A count to an *ex* command has been historically corrected to be no greater than the last line in a file; for example, in a five-line file, the command **1,6print** would fail, but the command **1print300** would succeed. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, the use of flags in *ex* commands could be obscure. General historical practice was as described by IEEE Std 1003.1-2001, but there were some special cases. For instance, the **list**, **number**, and **print** commands ignored trailing address offsets; for example, **3p +++#** would display line 3, and 3 would be the current line after the execution of the command. The **open** and **visual** commands ignored both the trailing offsets and the trailing flags. Also, flags specified to the **open** and **visual** commands interacted badly with the **list** edit option, and setting and then unsetting it during the open/visual session would cause *vi* to stop displaying lines in the specified format. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit any of these exceptions to the general rule.

IEEE Std 1003.1-2001 uses the word *copy* in several places when discussing buffers. This is not intended to imply implementation.

Historically, *ex* users could not specify numeric buffers because of the ambiguity this would cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a *count*. IEEE Std 1003.1-2001 requires conformance to historical practice by default, but does not preclude extensions.

Historically, the contents of the unnamed buffer were frequently discarded after commands that did not explicitly affect it; for example, when using the **edit** command to switch files. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric buffers would not have changed. IEEE Std 1003.1-2001 requires conformance to historical practice. Numeric buffers are described in the *ex* utility in order to confine the description of buffers to a single location in IEEE Std 1003.1-2001.

The metacharacters that trigger shell expansion in *file* arguments match historical practice, as does the method for doing shell expansion. Implementations wishing to provide users with the flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit

15906 option.

15907 Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to  
 15908 do so; for example, `!date > /dev/null` does not require a screen refresh because the output of the  
 15909 UNIX *date* command requires only a single line of the screen. IEEE Std 1003.1-2001 requires that  
 15910 the screen be refreshed if it has been overwritten, but makes no requirements as to how an  
 15911 implementation should make that determination. Implementations may prompt and refresh the  
 15912 screen regardless.

### 15913 **Abbreviate**

15914 Historical practice was that characters that were entered as part of an abbreviation replacement  
 15915 were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions,  
 15916 and so on; that is, they were logically pushed onto the terminal input queue, and were not a  
 15917 simple replacement. IEEE Std 1003.1-2001 requires conformance to historical practice. Historical  
 15918 practice was that whenever a non-word character (that had not been escaped by a `<control>-V`)  
 15919 was entered after a word character, *vi* would check for abbreviations. The check was based on  
 15920 the type of the character entered before the word character of the word/non-word pair that  
 15921 triggered the check. The word character of the word/non-word pair that triggered the check and  
 15922 all characters entered before the trigger pair that were of that type were included in the check,  
 15923 with the exception of `<blank>s`, which always delimited the abbreviation.

15924 This means that, for the abbreviation to work, the *lhs* must end with a word character, there can  
 15925 be no transitions from word to non-word characters (or *vice versa*) other than between the last  
 15926 and next-to-last characters in the *lhs*, and there can be no `<blank>s` in the *lhs*. In addition,  
 15927 because of the historical quoting rules, it was impossible to enter a literal `<control>-V` in the *lhs*.  
 15928 IEEE Std 1003.1-2001 requires conformance to historical practice. Historical implementations did  
 15929 not inform users when abbreviations that could never be used were entered; implementations  
 15930 are strongly encouraged to do so.

15931 For example, the following abbreviations will work:

```
15932 :ab (p REPLACE
15933 :ab p REPLACE
15934 :ab ((p REPLACE
```

15935 The following abbreviations will not work:

```
15936 :ab (REPLACE
15937 :ab (pp REPLACE
```

15938 Historical practice is that words on the *vi* colon command line were subject to abbreviation  
 15939 expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev**  
 15940 command. Because there are implementations that do not do abbreviation expansion for the first  
 15941 argument to those commands, this is permitted, but not required, by IEEE Std 1003.1-2001.  
 15942 However, the following sequence:

```
15943 :ab foo bar
15944 :ab foo baz
```

15945 resulted in the addition of an abbreviation of "baz" for the string "bar" in historical *ex/vi*, and  
 15946 the sequence:

```
15947 :ab foo1 bar
15948 :ab foo2 bar
15949 :unabbreviate foo2
```

15950 deleted the abbreviation "foo1", not "foo2". These behaviors are not permitted by  
 15951 IEEE Std 1003.1-2001 because they clearly violate the expectations of the user.

15952 It was historical practice that <control>-V, not backslash, characters be interpreted as escaping  
 15953 subsequent characters in the **abbreviate** command. IEEE Std 1003.1-2001 requires conformance  
 15954 to historical practice; however, it should be noted that an abbreviation containing a <blank> will  
 15955 never work.

## 15956 **Append**

15957 Historically, any text following a vertical-line command separator after an **append**, **change**, or  
 15958 **insert** command became part of the insert text. For example, in the command:

15959 :g/pattern/append|stuff1

15960 a line containing the text "stuff1" would be appended to each line matching pattern. It was  
 15961 also historically valid to enter:

15962 :append|stuff1

15963 stuff2

15964 .

15965 and the text on the *ex* command line would be appended along with the text inserted after it.  
 15966 There was an historical bug, however, that the user had to enter two terminating lines (the ' .'   
 15967 lines) to terminate text input mode in this case. IEEE Std 1003.1-2001 requires conformance to  
 15968 historical practice, but disallows the historical need for multiple terminating lines.

## 15969 **Change**

15970 See the RATIONALE for the **append** command. Historical practice for cursor positioning after  
 15971 the change command when no text is input, is as described in IEEE Std 1003.1-2001. However,  
 15972 one System V implementation is known to have been modified such that the cursor is positioned  
 15973 on the first address specified, and not on the line before the first address. IEEE Std 1003.1-2001  
 15974 disallows this modification for consistency.

15975 Historically, the **change** command did not support buffer arguments, although some  
 15976 implementations allow the specification of an optional buffer. This behavior is neither required  
 15977 nor disallowed by IEEE Std 1003.1-2001.

## 15978 **Change Directory**

15979 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as  
 15980 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have  
 15981 ' .' or " ." as their first component. Elements in the **cdpath** edit option are colon-separated.  
 15982 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment  
 15983 variable. This feature was not included in IEEE Std 1003.1-2001 because it does not exist in any  
 15984 of the implementations considered historical practice.

## 15985 **Copy**

15986 Historical implementations of *ex* permitted copies to lines inside of the specified range; for  
 15987 example, **:2,5copy3** was a valid command. IEEE Std 1003.1-2001 requires conformance to  
 15988 historical practice.

## 15989 Delete

15990 IEEE Std 1003.1-2001 requires support for the historical parsing of a **delete** command followed  
15991 by flags, without any intervening <blank>s. For example:

15992 **1dp** Deletes the first line and prints the line that was second.

15993 **1delep** As for **1dp**.

15994 **1d** Deletes the first line, saving it in buffer *p*.

15995 **1d p1l** (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was  
15996 second.

## 15997 Edit

15998 Historically, any *ex* command could be entered as a *+command* argument to the **edit** command,  
15999 although some (for example, **insert** and **append**) were known to confuse historical  
16000 implementations. For consistency and simplicity of specification, IEEE Std 1003.1-2001 requires  
16001 that any command be supported as an argument to the **edit** command.

16002 Historically, the command argument was executed with the current line set to the last line of the  
16003 file, regardless of whether the **edit** command was executed from visual mode or not.  
16004 IEEE Std 1003.1-2001 requires conformance to historical practice.

16005 Historically, the *+command* specified to the **edit** and **next** commands was delimited by the first  
16006 <blank>, and there was no way to quote them. For consistency, IEEE Std 1003.1-2001 requires  
16007 that the usual *ex* backslash quoting be provided.

16008 Historically, specifying the *+command* argument to the edit command required a filename to be  
16009 specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of  
16010 specification, IEEE Std 1003.1-2001 does not permit this usage to fail for that reason.

16011 Historically, only the cursor position of the last file edited was remembered by the editor.  
16012 IEEE Std 1003.1-2001 requires that this be supported; however, implementations are permitted to  
16013 remember and restore the cursor position for any file previously edited.

## 16014 File

16015 Historical versions of the *ex* editor **file** command displayed a current line and number of lines in  
16016 the edit buffer of 0 when the file was empty, while the *vi* <control>-G command displayed a  
16017 current line and number of lines in the edit buffer of 1 in the same situation.  
16018 IEEE Std 1003.1-2001 does not permit this discrepancy, instead requiring that a message be  
16019 displayed indicating that the file is empty.

## 16020 Global

16021 The two-pass operation of the **global** and **v** commands is not intended to imply implementation,  
16022 only the required result of the operation.

16023 The current line and column are set as specified for the individual *ex* commands. This  
16024 requirement is cumulative; that is, the current line and column must track across all the  
16025 commands executed by the **global** or **v** commands.



**Insert**

See the RATIONALE for the **append** command.

Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer was empty. IEEE Std 1003.1-2001 requires that this command behave consistently with the **append** command.

**Join**

The action of the **join** command in relation to the special characters is only defined for the POSIX locale because the correct amount of white space after a period varies; in Japanese none is required, in French only a single space, and so on.

**List**

The historical output of the **list** command was potentially ambiguous. The standard developers believed correcting this to be more important than adhering to historical practice, and IEEE Std 1003.1-2001 requires unambiguous output.

**Map**

Historically, command mode maps only applied to command names; for example, if the character 'x' was mapped to 'y', the command **fx** searched for the 'x' character, not the 'y' character. IEEE Std 1003.1-2001 requires this behavior. Historically, entering <control>-V as the first character of a **vi** command was an error. Several implementations have extended the semantics of **vi** such that <control>-V means that the subsequent command character is not mapped. This is permitted, but not required, by IEEE Std 1003.1-2001. Regardless, using <control>-V to escape the second or later character in a sequence of characters that might match a **map** command, or any character in text input mode, is historical practice, and stops the entered keys from matching a map. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored the map. IEEE Std 1003.1-2001 requires that the mapped digits not be ignored.

The historical implementation of the **map** command did not permit **map** commands that were more than a single character in length if the first character was printable. This behavior is permitted, but not required, by IEEE Std 1003.1-2001.

Historically, mapped characters were remapped unless the **remap** edit option was not set, or the prefix of the mapped characters matched the mapping characters; for example, in the **map**:

```
:map ab abcd
```

the characters "ab" were used as is and were not remapped, but the characters "cd" were mapped if appropriate. This can cause infinite loops in the **vi** mapping mechanisms. IEEE Std 1003.1-2001 requires conformance to historical practice, and that such loops be interruptible.

Text input maps had the same problems with expanding the *lhs* for the **ex map!** and **unmap!** command as did the **ex abbreviate** and **unabbreviate** commands. See the RATIONALE for the **ex abbreviate** command. IEEE Std 1003.1-2001 requires similar modification of some historical practice for the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate** commands.

Historically, **maps** that were subsets of other **maps** behaved differently depending on the order in which they were defined. For example:

```
16068 :map! ab short
16069 :map! abc long
```

16070 would always translate the characters "ab" to "short", regardless of how fast the characters  
16071 "abc" were entered. If the entry order was reversed:

```
16072 :map! abc long
16073 :map! ab short
```

16074 the characters "ab" would cause the editor to pause, waiting for the completing 'c' character,  
16075 and the characters might never be mapped to "short". For consistency and simplicity of  
16076 specification, IEEE Std 1003.1-2001 requires that the shortest match be used at all times.

16077 The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified  
16078 because the timing capabilities of systems are often inexact and variable, and it may depend on  
16079 other factors such as the speed of the connection. The time should be long enough for the user to  
16080 be able to complete the sequence, but not long enough for the user to have to wait. Some  
16081 implementations of *vi* have added a **keytime** option, which permits users to set the number of  
16082 0,1 seconds the editor waits for the completing characters. Because mapped terminal function  
16083 and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input  
16084 mode, **maps** starting with <ESC> characters are generally exempted from this timeout period,  
16085 or, at least timed out differently.

## 16086 Mark

16087 Historically, users were able to set the "previous context" marks explicitly. In addition, the *ex*  
16088 commands " and " and the *vi* commands ", ", ", and " all referred to the same mark. In addition,  
16089 the previous context marks were not set if the command, with which the address setting the  
16090 mark was associated, failed. IEEE Std 1003.1-2001 requires conformance to historical practice.  
16091 Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the  
16092 change was undone. IEEE Std 1003.1-2001 requires conformance to historical practice.

16093 The description of the special events that set the ' and ' marks matches historical practice. For  
16094 example, historically the command */a,/b/* did not set the ' and ' marks, but the command  
16095 */a,/b/delete* did.

## 16096 Next

16097 Historically, any *ex* command could be entered as a *+command* argument to the **next** command,  
16098 although some (for example, **insert** and **append**) were known to confuse historical  
16099 implementations. IEEE Std 1003.1-2001 requires that any command be permitted and that it  
16100 behave as specified. The **next** command can accept more than one file, so usage such as:

```
16101 next `ls [abc] `
```

16102 is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect  
16103 only one filename.

16104 Historically, the **next** command behaved differently from the **:rewind** command in that it  
16105 ignored the force flag if the **autowrite** flag was set. For consistency, IEEE Std 1003.1-2001 does  
16106 not permit this behavior.

16107 Historically, the **next** command positioned the cursor as if the file had never been edited before,  
16108 regardless. IEEE Std 1003.1-2001 does not permit this behavior, for consistency with the **edit**  
16109 command.

16110 Implementations wanting to provide a counterpart to the **next** command that edited the  
16111 previous file have used the command **prev[ious]**, which takes no *file* argument.

16112 IEEE Std 1003.1-2001 does not require this command.

### 16113 **Open**

16114 Historically, the **open** command would fail if the **open** edit option was not set.  
 16115 IEEE Std 1003.1-2001 does not mention the **open** edit option and does not require this behavior.  
 16116 Some historical implementations do not permit entering open mode from open or visual mode,  
 16117 only from **ex** mode. For consistency, IEEE Std 1003.1-2001 does not permit this behavior.

16118 Historically, entering open mode from the command line (that is, **vi +open**) resulted in  
 16119 anomalous behaviors; for example, the **ex** file and **set** commands, and the **vi** command  
 16120 <control>-G did not work. For consistency, IEEE Std 1003.1-2001 does not permit this behavior.

16121 Historically, the **open** command only permitted ' / ' characters to be used as the search pattern  
 16122 delimiter. For consistency, IEEE Std 1003.1-2001 requires that the search delimiters used by the **s**,  
 16123 **global**, and **v** commands be accepted as well.

### 16124 **Preserve**

16125 The **preserve** command does not historically cause the file to be considered unmodified for the  
 16126 purposes of future commands that may exit the editor. IEEE Std 1003.1-2001 requires  
 16127 conformance to historical practice.

16128 Historical documentation stated that mail was not sent to the user when preserve was executed;  
 16129 however, historical implementations did send mail in this case. IEEE Std 1003.1-2001 requires  
 16130 conformance to the historical implementations.

### 16131 **Print**

16132 The writing of NUL by the **print** command is not specified as a special case because the standard  
 16133 developers did not want to require **ex** to support NUL characters. Historically, characters were  
 16134 displayed using the ARPA standard mappings, which are as follows:

- 16135 1. Printable characters are left alone.
- 16136 2. Control characters less than \177 are represented as ' ^ ' followed by the character offset  
 16137 from the ' @ ' character in the ASCII map; for example, \007 is represented as ' ^G '.
- 16138 3. \177 is represented as ' ^ ' followed by ' ? '.

16139 The display of characters having their eighth bit set was less standard. Existing implementations  
 16140 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their  
 16141 eighth bit set as the two characters "M-" followed by the seven-bit display as described above.)  
 16142 The latter probably has the best claim to historical practice because it was used for the **-v** option  
 16143 of 4 BSD and 4 BSD-derived versions of the **cat** utility since 1980.

16144 No specific display format is required by IEEE Std 1003.1-2001.

16145 Explicit dependence on the ASCII character set has been avoided where possible, hence the use  
 16146 of the phrase an "implementation-defined multi-character sequence" for the display of non-  
 16147 printable characters in preference to the historical usage of, for instance, " ^I " for the <tab>.  
 16148 Implementations are encouraged to conform to historical practice in the absence of any strong  
 16149 reason to diverge.

16150 Historically, all **ex** commands beginning with the letter ' p ' could be entered using capitalized  
 16151 versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command  
 16152 names. IEEE Std 1003.1-2001 permits, but does not require, this historical practice because  
 16153 capital forms of the commands are used by some implementations for other purposes.

16154 **Put**

16155 Historically, an **ex put** command, executed from open or visual mode, was the same as the open  
16156 or visual mode **P** command, if the buffer was named and was cut in character mode, and the  
16157 same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer  
16158 was the source of the text, the entire line from which the text was taken was usually **put**, and the  
16159 buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior.  
16160 In addition, using the **Q** command to switch into **ex** mode, and then doing a **put** often resulted in  
16161 errors as well, such as appending text that was unrelated to the (supposed) contents of the  
16162 buffer. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit  
16163 these behaviors. All **ex put** commands are required to operate in line mode, and the contents of  
16164 the buffers are not altered by changing the mode of the editor.

16165 **Read**

16166 Historically, an **ex read** command executed from open or visual mode, executed in an empty file,  
16167 left an empty line as the first line of the file. For consistency and simplicity of specification,  
16168 IEEE Std 1003.1-2001 does not permit this behavior. Historically, a **read** in open or visual mode  
16169 from a program left the cursor at the last line read in, not the first. For consistency,  
16170 IEEE Std 1003.1-2001 does not permit this behavior.

16171 Historical implementations of **ex** were unable to undo **read** commands that read from the output  
16172 of a program. For consistency, IEEE Std 1003.1-2001 does not permit this behavior.

16173 Historically, the **ex** and **vi** message after a successful **read** or **write** command specified  
16174 “characters”, not “bytes”. IEEE Std 1003.1-2001 requires that the number of bytes be displayed,  
16175 not the number of characters, because it may be difficult in multi-byte implementations to  
16176 determine the number of characters read. Implementations are encouraged to clarify the  
16177 message displayed to the user.

16178 Historically, reads were not permitted on files other than type regular, except that FIFO files  
16179 could be read (probably only because they did not exist when **ex** and **vi** were originally written).  
16180 Because the historical **ex** evaluated **read!** and **read !** equivalently, there can be no optional way  
16181 to force the read. IEEE Std 1003.1-2001 permits, but does not require, this behavior.

16182 **Recover**

16183 Some historical implementations of the editor permitted users to recover the edit buffer contents  
16184 from a previous edit session, and then exit without saving those contents (or explicitly  
16185 discarding them). The intent of IEEE Std 1003.1-2001 in requiring that the edit buffer be treated  
16186 as already modified is to prevent this user error.

16187 **Rewind**

16188 Historical implementations supported the **rewind** command when the user was editing the first  
16189 file in the list; that is, the file that the **rewind** command would edit. IEEE Std 1003.1-2001  
16190 requires conformance to historical practice.

**Substitute**

Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the last regular expression used in any command as the pattern, the same as the **~** command. The **r** option is not required by IEEE Std 1003.1-2001. Historically, the **c** and **g** options were toggled; for example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

The tilde command is often used to replace the last search RE. For example, in the sequence:

```
s/red/blue/
/green
~
```

the **~** command is equivalent to:

```
s/green/blue/
```

Historically, *ex* accepted all of the following forms:

```
s/abc/def/
s/abc/def
s/abc/
s/abc
```

IEEE Std 1003.1-2001 requires conformance to this historical practice.

The **s** command presumes that the **^** character only occupies a single column in the display. Much of the *ex* and *vi* specification presumes that the **<space>** only occupies a single column in the display. There are no known character sets for which this is not true.

Historically, the final column position for the substitute commands was based on previous column movements; a search for a pattern followed by a substitution would leave the column position unchanged, while a **0** command followed by a substitution would change the column position to the first non-**<blank>**. For consistency and simplicity of specification, IEEE Std 1003.1-2001 requires that the final column position always be set to the first non-**<blank>**.

**Set**

Historical implementations redisplayed all of the options for each occurrence of the **all** keyword. IEEE Std 1003.1-2001 permits, but does not require, this behavior.

**Tag**

No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry. Historical practice has been to look for the path found in the **tags** file, based on the current directory. A useful extension found in some implementations is to look based on the directory containing the tags file that held the entry, as well. No requirement is made as to which reference for the tag in the tags file is used. This is deliberate, in order to permit extensions such as multiple entries in a tags file for a tag.

Because users often specify many different tags files, some of which need not be relevant or exist at any particular time, IEEE Std 1003.1-2001 requires that error messages about problem tags files be displayed only if the requested tag is not found, and then, only once for each time that the **tag** edit option is changed.

The requirement that the current edit buffer be unmodified is only necessary if the file indicated by the tag entry is not the same as the current file (as defined by the current pathname).

Historically, the file would be reloaded if the filename had changed, as well as if the filename was different from the current pathname. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior, requiring that the name be the only factor in the decision.

Historically, *vi* only searched for tags in the current file from the current cursor to the end of the file, and therefore, if the **wrapscan** option was not set, tags occurring before the current cursor were not found. IEEE Std 1003.1-2001 considers this a bug, and implementations are required to search for the first occurrence in the file, regardless.

#### 16242 **Undo**

The **undo** description deliberately uses the word “modified”. The **undo** command is not intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**, or **recover**.

Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and sometimes, in the presence of maps, placing the cursor on the last line added or changed instead of the first. IEEE Std 1003.1-2001 requires a simplified behavior for consistency and simplicity of specification.

#### 16251 **Version**

The **version** command cannot be exactly specified since there is no widely-accepted definition of what the version information should contain. Implementations are encouraged to do something reasonably intelligent.

#### 16255 **Write**

Historically, the *ex* and *vi* message after a successful **read** or **write** command specified “characters”, not “bytes”. IEEE Std 1003.1-2001 requires that the number of bytes be displayed, not the number of characters because it may be difficult in multi-byte implementations to determine the number of characters written. Implementations are encouraged to clarify the message displayed to the user.

Implementation-defined tests are permitted so that implementations can make additional checks; for example, for locks or file modification times.

Historically, attempting to append to a nonexistent file caused an error. It has been left unspecified in IEEE Std 1003.1-2001 to permit implementations to let the **write** succeed, so that the append semantics are similar to those of the historical *csh*.

Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around dealing with “empty” files was to always have a line in the edit buffer, no matter what, it wrote them as files of a single, empty line. IEEE Std 1003.1-2001 does not permit this behavior.

Historically, *ex* restored standard output and standard error to their values as of when *ex* was invoked, before writes to programs were performed. This could disturb the terminal configuration as well as be a security issue for some terminals. IEEE Std 1003.1-2001 does not permit this, requiring that the program output be captured and displayed as if by the *ex* **print** command.

**Adjust Window**

Historically, the line count was set to the value of the **scroll** option if the type character was end-of-file. This feature was broken on most historical implementations long ago, however, and is not documented anywhere. For this reason, IEEE Std 1003.1-2001 is resolutely silent.

Historically, the **z** command was <blank>-sensitive and **z +** and **z -** did different things than **z+** and **z-** because the type could not be distinguished from a flag. (The commands **z .** and **z =** were historically invalid.) IEEE Std 1003.1-2001 requires conformance to this historical practice.

Historically, the **z** command was further <blank>-sensitive in that the *count* could not be <blank>-delimited; for example, the commands **z= 5** and **z- 5** were also invalid. Because the *count* is not ambiguous with respect to either the type character or the flags, this is not permitted by IEEE Std 1003.1-2001.

**Escape**

Historically, *ex* filter commands only read the standard output of the commands, letting standard error appear on the terminal as usual. The *vi* utility, however, read both standard output and standard error. IEEE Std 1003.1-2001 requires the latter behavior for both *ex* and *vi*, for consistency.

**Shift Left and Shift Right**

Historically, it was possible to add shift characters to increase the effect of the command; for example, <<< outdented (or >>> indented) the lines 3 levels of indentation instead of the default 1. IEEE Std 1003.1-2001 requires conformance to historical practice.

**<control>-D**

Historically, the <control>-D command erased the prompt, providing the user with an unbroken presentation of lines from the edit buffer. This is not required by IEEE Std 1003.1-2001; implementations are encouraged to provide it if possible. Historically, the <control>-D command took, and then ignored, a *count*. IEEE Std 1003.1-2001 does not permit this behavior.

**Write Line Number**

Historically, the *ex* = command, when executed in *ex* mode in an empty edit buffer, reported 0, and from open or visual mode, reported 1. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

**Execute**

Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**, **insert**, and **change**) in executed buffers. IEEE Std 1003.1-2001 does not permit this exclusion for consistency.

Historically, the logical contents of the buffer being executed did not change if the buffer itself were modified by the commands being executed; that is, buffer execution did not support self-modifying code. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, the @ command took a range of lines, and the @ buffer was executed once per line, with the current line ( ' . ' ) set to each specified line. IEEE Std 1003.1-2001 requires conformance to historical practice.

Some historical implementations did not notice if errors occurred during buffer execution. This, coupled with the ability to specify a range of lines for the *ex* @ command, makes it trivial to cause them to drop **core**. IEEE Std 1003.1-2001 requires that implementations stop buffer

16316 execution if any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer  
16317 itself are replaced (for example, the buffer executes the **ex :edit** command).

## 16318 **Regular Expressions in ex**

16319 Historical practice is that the characters in the replacement part of the last **s** command—that is,  
16320 those matched by entering a '~' in the regular expression—were not further expanded by the  
16321 regular expression engine. So, if the characters contained the string "a.," they would match  
16322 'a' followed by ".," and not 'a' followed by any character. IEEE Std 1003.1-2001 requires  
16323 conformance to historical practice.

## 16324 **Edit Options in ex**

16325 The following paragraphs describe the historical behavior of some edit options that were not, for  
16326 whatever reason, included in IEEE Std 1003.1-2001. Implementations are strongly encouraged to  
16327 only use these names if the functionality described here is fully supported.

16328 **extended** The **extended** edit option has been used in some implementations of *vi* to provide  
16329 extended regular expressions instead of basic regular expressions. This option was  
16330 omitted from IEEE Std 1003.1-2001 because it is not widespread historical practice.

16331 **flash** The **flash** edit option historically caused the screen to flash instead of beeping on  
16332 error. This option was omitted from IEEE Std 1003.1-2001 because it is not found in  
16333 some historical implementations.

16334 **hardtabs** The **hardtabs** edit option historically defined the number of columns between  
16335 hardware tab settings. This option was omitted from IEEE Std 1003.1-2001 because  
16336 it was believed to no longer be generally useful.

16337 **modeline** The **modeline** (sometimes named **modelines**) edit option historically caused *ex* or  
16338 *vi* to read the five first and last lines of the file for editor commands. This option is  
16339 a security problem, and vendors are strongly encouraged to delete it from  
16340 historical implementations.

16341 **open** The **open** edit option historically disallowed the *ex* **open** and **visual** commands.  
16342 This edit option was omitted because these commands are required by  
16343 IEEE Std 1003.1-2001.

16344 **optimize** The **optimize** edit option historically expedited text throughput by setting the  
16345 terminal to not do automatic <carriage-return>s when printing more than one  
16346 logical line of output. This option was omitted from IEEE Std 1003.1-2001 because  
16347 it was intended for terminals without addressable cursors, which are rarely, if ever,  
16348 still used.

16349 **ruler** The **ruler** edit option has been used in some implementations of *vi* to present a  
16350 current row/column ruler for the user. This option was omitted from  
16351 IEEE Std 1003.1-2001 because it is not widespread historical practice.

16352 **sourceany** The **sourceany** edit option historically caused *ex* or *vi* to source start-up files that  
16353 were owned by users other than the user running the editor. This option is a  
16354 security problem, and vendors are strongly encouraged to remove it from their  
16355 implementations.

16356 **timeout** The **timeout** edit option historically enabled the (now standard) feature of only  
16357 waiting for a short period before returning keys that could be part of a macro. This  
16358 feature was omitted from IEEE Std 1003.1-2001 because its behavior is now  
16359 standard, it is not widely useful, and it was rarely documented.



|       |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 16360 | <b>verbose</b>        | The <b>verbose</b> edit option has been used in some implementations of <i>vi</i> to cause <i>vi</i> to output error messages for common errors; for example, attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical <i>vi</i> only alerted the terminal and presented no message for such errors. The historical editor option <b>terse</b> did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from IEEE Std 1003.1-2001 because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users. |
| 16361 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16362 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16363 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16364 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16365 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16366 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16367 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16368 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16369 | <b>wraplen</b>        | The <b>wraplen</b> edit option has been used in some implementations of <i>vi</i> to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This option was omitted from IEEE Std 1003.1-2001 because it is not widespread historical practice; however, implementors are encouraged to use it if they add this functionality.                                                                                                                                                                                                                                                                                  |
| 16370 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16371 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16372 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16373 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16374 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16375 | <b>autoindent, ai</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16376 |                       | Historically, the command <b>0a</b> did not do any autoindentation, regardless of the current indentation of line 1. IEEE Std 1003.1-2001 requires that any indentation present in line 1 be used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 16377 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16378 | <b>autoprint, ap</b>  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16379 |                       | Historically, the <b>autoprint</b> edit option was not completely consistent or based solely on modifications to the edit buffer. Exceptions were the <b>read</b> command (when reading from a file, but not from a filter), the <b>append</b> , <b>change</b> , <b>insert</b> , <b>global</b> , and <b>v</b> commands, all of which were not affected by <b>autoprint</b> , and the <b>tag</b> command, which was affected by <b>autoprint</b> . IEEE Std 1003.1-2001 requires conformance to historical practice.                                                                                                                                                                                                                  |
| 16380 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16381 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16382 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16383 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16384 |                       | Historically, the <b>autoprint</b> option only applied to the last of multiple commands entered using vertical-bar delimiters; for example, <b>delete</b> <newline> was affected by <b>autoprint</b> , but <b>delete version</b> <newline> was not. IEEE Std 1003.1-2001 requires conformance to historical practice.                                                                                                                                                                                                                                                                                                                                                                                                                |
| 16385 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16386 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16387 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16388 | <b>autowrite, aw</b>  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16389 |                       | Appending the '!' character to the <i>ex</i> <b>next</b> command to avoid performing an automatic write was not supported in historical implementations. IEEE Std 1003.1-2001 requires that the behavior match the other <i>ex</i> commands for consistency.                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 16390 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16391 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16392 | <b>ignorecase, ic</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16393 |                       | Historical implementations of case-insensitive matching (the <b>ignorecase</b> edit option) lead to counterintuitive situations when uppercase characters were used in range expressions. Historically, the process was as follows:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 16394 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16395 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16396 |                       | 1. Take a line of text from the edit buffer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 16397 |                       | 2. Convert uppercase to lowercase in text line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16398 |                       | 3. Convert uppercase to lowercase in regular expressions, except in character class specifications.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 16399 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 16400 |                       | 4. Match regular expressions against text.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 16401 |                       | This would mean that, with <b>ignorecase</b> in effect, the text:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

16402 The cat sat on the mat  
 16403 would be matched by  
 16404 /^the/  
 16405 but not by:  
 16406 /^ [A-Z] he/

16407 For consistency with other commands implementing regular expressions, IEEE Std 1003.1-2001  
 16408 does not permit this behavior.

#### 16409 **paragraphs, para**

16410 The ISO POSIX-2:1993 standard made the default **paragraphs** and **sections** edit options  
 16411 implementation-defined, arguing they were historically oriented to the UNIX system *troff* text  
 16412 formatter, and a “portable user” could use the { }, [[, ]], (, and ) commands in open or visual  
 16413 mode and have the cursor stop in unexpected places. IEEE Std 1003.1-2001 specifies their values  
 16414 in the POSIX locale because the unusual grouping (they only work when grouped into two  
 16415 characters at a time) means that they cannot be used for general-purpose movement, regardless.

#### 16416 **readonly**

16417 Implementations are encouraged to provide the best possible information to the user as to the  
 16418 read-only status of the file, with the exception that they should not consider the current special  
 16419 privileges of the process. This provides users with a safety net because they must force the  
 16420 overwrite of read-only files, even when running with additional privileges.

16421 The **readonly** edit option specification largely conforms to historical practice. The only  
 16422 difference is that historical implementations did not notice that the user had set the **readonly**  
 16423 edit option in cases where the file was already marked read-only for some reason, and would  
 16424 therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were  
 16425 replaced. This behavior is disallowed by IEEE Std 1003.1-2001.

#### 16426 **report**

16427 The requirement that lines copied to a buffer interact differently than deleted lines is historical  
 16428 practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be  
 16429 written, but 4 lines must be copied before a report is written.

16430 The requirement that the *ex* **global**, **v**, **open**, **undo**, and **visual** commands present reports based  
 16431 on the total number of lines added or deleted during the command execution, and that  
 16432 commands executed by the **global** and **v** commands not present reports, is historical practice.  
 16433 IEEE Std 1003.1-2001 extends historical practice by requiring that buffer execution be treated  
 16434 similarly. The reasons for this are two-fold. Historically, only the report by the last command  
 16435 executed from the buffer would be seen by the user, as each new report would overwrite the  
 16436 last. In addition, the standard developers believed that buffer execution had more in common  
 16437 with **global** and **v** commands than it did with other *ex* commands, and should behave similarly,  
 16438 for consistency and simplicity of specification.

**showmatch, sm**

The length of time the cursor spends on the matching character is unspecified because the timing capabilities of systems are often inexact and variable. The time should be long enough for the user to notice, but not long enough for the user to become annoyed. Some implementations of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals the cursor pauses on the matching character.

**showmode**

The **showmode** option has been used in some historical implementations of *ex* and *vi* to display the current editing mode when in open or visual mode. The editing modes have generally included “command” and “input”, and sometimes other modes such as “replace” and “change”. The string was usually displayed on the bottom line of the screen at the far right-hand corner. In addition, a preceding ‘\*’ character often denoted whether the contents of the edit buffer had been modified. The latter display has sometimes been part of the **showmode** option, and sometimes based on another option. This option was not available in the 4 BSD historical implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is required by IEEE Std 1003.1-2001.

The **smd** shorthand for the **showmode** option was not present in all historical implementations of the editor. IEEE Std 1003.1-2001 requires it, for consistency.

Not all historical implementations of the editor displayed a mode string for command mode, differentiating command mode from text input mode by the absence of a mode string. IEEE Std 1003.1-2001 permits this behavior for consistency with historical practice, but implementations are encouraged to provide a display string for both modes.

**slowopen**

Historically the **slowopen** option was automatically set if the terminal baud rate was less than 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen** option had two effects. First, when inserting characters in the middle of a line, characters after the cursor would not be pushed ahead, but would appear to be overwritten. Second, when creating a new line of text, lines after the current line would not be scrolled down, but would appear to be overwritten. In both cases, ending text input mode would cause the screen to be refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently intelligent caused the editor to ignore the **slowopen** option. IEEE Std 1003.1-2001 permits most historical behavior, extending historical practice to require **slowopen** behaviors if the edit option is set by the user.

**tags**

The default path for tags files is left unspecified as implementations may have their own **tags** implementations that do not correspond to the historical ones. The default **tags** option value should probably at least include the file **./tags**.

**term**

Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial terminal information was loaded. This is permitted by IEEE Std 1003.1-2001; however, implementations are encouraged to permit the user to modify their terminal type at any time.

**terse**

Historically, the **terse** edit option optionally provided a shorter, less descriptive error message, for some error messages. This is permitted, but not required, by IEEE Std 1003.1-2001. Historically, most common visual mode errors (for example, trying to move the cursor past the end of a line) did not result in an error message, but simply alerted the terminal. Implementations wishing to provide messages for novice users are urged to do so based on the **edit** option **verbose**, and not **terse**.

**window**

In historical implementations, the default for the **window** edit option was based on the baud rate as follows:

1. If the baud rate was less than 1200, the **edit** option **w300** set the window value; for example, the line:

```
set w300=12
```

would set the window option to 12 if the baud rate was less than 1200.

2. If the baud rate was equal to 1200, the **edit** option **w1200** set the window value.
3. If the baud rate was greater than 1200, the **edit** option **w9600** set the window value.

The **w300**, **w1200**, and **w9600** options do not appear in IEEE Std 1003.1-2001 because of their dependence on specific baud rates.

In historical implementations, the size of the window displayed by various commands was related to, but not necessarily the same as, the **window** edit option. For example, the size of the window was set by the *ex* command **visual 10**, but it did not change the value of the **window** edit option. However, changing the value of the **window** edit option did change the number of lines that were displayed when the screen was repainted. IEEE Std 1003.1-2001 does not permit this behavior in the interests of consistency and simplicity of specification, and requires that all commands that change the number of lines that are displayed do it by setting the value of the **window** edit option.

**wrapmargin, wm**

Historically, the **wrapmargin** option did not affect maps inserting characters that also had associated *counts*; for example **:map K 5aABC DEF**. Unfortunately, there are widely used maps that depend on this behavior. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

Historically, **wrapmargin** was calculated using the column display width of all characters on the screen. For example, an implementation using "^I" to represent <tab>s when the **list** edit option was set, where '^' and 'I' each took up a single column on the screen, would calculate the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option similarly changed the effective length of the line as well. IEEE Std 1003.1-2001 requires conformance to historical practice.

16517 **FUTURE DIRECTIONS**

16518 None.

16519 **SEE ALSO**

16520 Section 2.9.1.1 (on page 48), *ctags*, *ed*, *sed*, *sh*, *stty*, *vi*, the System Interfaces volume of  
16521 IEEE Std 1003.1-2001, *access()*

16522 **CHANGE HISTORY**

16523 First released in Issue 2.

16524 **Issue 5**

16525 The FUTURE DIRECTIONS section is added.

16526 **Issue 6**

16527 This utility is marked as part of the User Portability Utilities option.

16528 The obsolescent SYNOPSIS is removed, removing the *+command* and *-* options.

16529 The following new requirements on POSIX implementations derive from alignment with the  
16530 Single UNIX Specification:

16531 • In the **map** command description, the sequence *#digit* is added.

16532 • The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.

16533 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This  
16534 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,  
16535 #55, #56, #57, #61, #62, #63, #64, #65, and #78.

16536 The **-I** option is removed.

16537 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/23 is applied, correcting a URL.

16538 **NAME**

16539           expand — convert tabs to spaces

16540 **SYNOPSIS**16541 UP       expand [-t *tablist*] [*file* ...]

16542

16543 **DESCRIPTION**

16544       The *expand* utility shall write files or the standard input to the standard output with <tab>s  
 16545       replaced with one or more <space>s needed to pad to the next tab stop. Any <backspace>s shall  
 16546       be copied to the output and cause the column position count for tab stop calculations to be  
 16547       decremented; the column position count shall not be decremented below zero.

16548 **OPTIONS**

16549       The *expand* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 16550       12.2, Utility Syntax Guidelines.

16551       The following option shall be supported:

16552       -t *tablist*   Specify the tab stops. The application shall ensure that the argument *tablist*  
 16553       consists of either a single positive decimal integer or a list of tabstops. If a single  
 16554       number is given, tabs shall be set that number of column positions apart instead of  
 16555       the default 8.

16556       If a list of tabstops is given, the application shall ensure that it consists of a list of  
 16557       two or more positive decimal integers, separated by <blank>s or commas, in  
 16558       ascending order. The tabs shall be set at those specific column positions. Each tab  
 16559       stop *N* shall be an integer value greater than zero, and the list is in strictly  
 16560       ascending order. This is taken to mean that, from the start of a line of output,  
 16561       tabbing to position *N* shall cause the next character output to be in the (*N*+1)th  
 16562       column position on that line.

16563       In the event of *expand* having to process a <tab> at a position beyond the last of  
 16564       those specified in a multiple tab-stop list, the <tab> shall be replaced by a single  
 16565       <space> in the output.

16566 **OPERANDS**

16567       The following operand shall be supported:

16568       *file*           The pathname of a text file to be used as input.

16569 **STDIN**

16570       See the INPUT FILES section.

16571 **INPUT FILES**

16572       Input files shall be text files.

16573 **ENVIRONMENT VARIABLES**

16574       The following environment variables shall affect the execution of *expand*:

16575       *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 16576       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 16577       Internationalization Variables for the precedence of internationalization variables  
 16578       used to determine the values of locale categories.)

16579       *LC\_ALL*          If set to a non-empty string value, override the values of all the other  
 16580       internationalization variables.

16581       *LC\_CTYPE*       Determine the locale for the interpretation of sequences of bytes of text data as  
 16582       characters (for example, single-byte as opposed to multi-byte characters in

16583 arguments and input files), the processing of <tab>s and <space>s, and for the  
 16584 determination of the width in column positions each character would occupy on  
 16585 an output device.

16586 **LC\_MESSAGES**

16587 Determine the locale that should be used to affect the format and contents of  
 16588 diagnostic messages written to standard error.

16589 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

16590 **ASYNCHRONOUS EVENTS**

16591 Default.

16592 **STDOUT**

16593 The standard output shall be equivalent to the input files with <tab>s converted into the  
 16594 appropriate number of <space>s.

16595 **STDERR**

16596 The standard error shall be used only for diagnostic messages.

16597 **OUTPUT FILES**

16598 None.

16599 **EXTENDED DESCRIPTION**

16600 None.

16601 **EXIT STATUS**

16602 The following exit values shall be returned:

16603 0 Successful completion

16604 >0 An error occurred.

16605 **CONSEQUENCES OF ERRORS**

16606 The *expand* utility shall terminate with an error message and non-zero exit status upon  
 16607 encountering difficulties accessing one of the *file* operands.

16608 **APPLICATION USAGE**

16609 None.

16610 **EXAMPLES**

16611 None.

16612 **RATIONALE**

16613 The *expand* utility is useful for preprocessing text files (before sorting, looking at specific  
 16614 columns, and so on) that contain <tab>s.

16615 See the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.103, Column Position.

16616 The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8  
 16617 mandates that *expand* shall accept the integers (within the single argument) separated using  
 16618 either commas or <blank>s.

16619 **FUTURE DIRECTIONS**

16620 None.

16621 **SEE ALSO**

16622 *tabs*, *unexpand*

16623 **CHANGE HISTORY**

16624 First released in Issue 4.

16625 **Issue 6**

16626 This utility is marked as part of the User Portability Utilities option.

16627 The APPLICATION USAGE section is added.

16628 The obsolescent SYNOPSIS is removed.

16629 The *LC\_CTYPE* environment variable description is updated to align with the IEEE P1003.2b  
16630 draft standard.

16631 The normative text is reworded to avoid use of the term “must” for application requirements.



16632 **NAME**16633        *expr* — evaluate arguments as an expression16634 **SYNOPSIS**16635        *expr operand*16636 **DESCRIPTION**16637        The *expr* utility shall evaluate an expression and write the result to standard output.16638 **OPTIONS**

16639        None.

16640 **OPERANDS**

16641        The single expression evaluated by *expr* shall be formed from the operands, as described in the  
 16642        EXTENDED DESCRIPTION section. The application shall ensure that each of the expression  
 16643        operator symbols:

16644        ( ) | &amp; = &gt; &gt;= &lt; &lt;= != + - \* / % :

16645        and the symbols *integer* and *string* in the table are provided as separate arguments to *expr*.16646 **STDIN**

16647        Not used.

16648 **INPUT FILES**

16649        None.

16650 **ENVIRONMENT VARIABLES**16651        The following environment variables shall affect the execution of *expr*:

16652        *LANG*        Provide a default value for the internationalization variables that are unset or null.  
 16653        (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 16654        Internationalization Variables for the precedence of internationalization variables  
 16655        used to determine the values of locale categories.)

16656        *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
 16657        internationalization variables.

16658        *LC\_COLLATE*    Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 16659        character collating elements within regular expressions and by the string  
 16660        comparison operators.  
 16661

16662        *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
 16663        characters (for example, single-byte as opposed to multi-byte characters in  
 16664        arguments) and the behavior of character classes within regular expressions.

16665        *LC\_MESSAGES*    Determine the locale that should be used to affect the format and contents of  
 16666        diagnostic messages written to standard error.  
 16667

16668 *XSI*        *NLSPATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

16669 **ASYNCHRONOUS EVENTS**

16670        Default.

16671 **STDOUT**

16672        The *expr* utility shall evaluate the expression and write the result, followed by a <newline>, to  
 16673        standard output.

16674 **STDERR**

16675 The standard error shall be used only for diagnostic messages.

16676 **OUTPUT FILES**

16677 None.

16678 **EXTENDED DESCRIPTION**

16679 The formation of the expression to be evaluated is shown in the following table. The symbols  
 16680 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the  
 16681 expression operator symbols (all separate arguments) by recursive application of the constructs  
 16682 described in the table. The expressions are listed in order of increasing precedence, with equal-  
 16683 precedence operators grouped between horizontal lines. All of the operators shall be left-  
 16684 associative.

| Expression                                                                                                                                                                                | Description                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>expr1</i>   <i>expr2</i>                                                                                                                                                               | Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> if it is not null; otherwise, zero.                                                                                                                                                                                                                                               |
| <i>expr1</i> & <i>expr2</i>                                                                                                                                                               | Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.                                                                                                                                                                                                                                                                                           |
| <i>expr1</i> = <i>expr2</i><br><i>expr1</i> > <i>expr2</i><br><i>expr1</i> >= <i>expr2</i><br><i>expr1</i> < <i>expr2</i><br><i>expr1</i> <= <i>expr2</i><br><i>expr1</i> != <i>expr2</i> | Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false.<br>Equal.<br>Greater than.<br>Greater than or equal.<br>Less than.<br>Less than or equal.<br>Not equal. |
| <i>expr1</i> + <i>expr2</i><br><i>expr1</i> − <i>expr2</i>                                                                                                                                | Addition of decimal integer-valued arguments.<br>Subtraction of decimal integer-valued arguments.                                                                                                                                                                                                                                                                                                          |
| <i>expr1</i> * <i>expr2</i><br><i>expr1</i> / <i>expr2</i><br><i>expr1</i> % <i>expr2</i>                                                                                                 | Multiplication of decimal integer-valued arguments.<br>Integer division of decimal integer-valued arguments, producing an integer result.<br>Remainder of integer division of decimal integer-valued arguments.                                                                                                                                                                                            |
| <i>expr1</i> : <i>expr2</i>                                                                                                                                                               | Matching expression; see below.                                                                                                                                                                                                                                                                                                                                                                            |
| ( <i>expr</i> )                                                                                                                                                                           | Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.                                                                                                                                                                                                                                                                                |
| <i>integer</i><br><i>string</i>                                                                                                                                                           | An argument consisting only of an (optional) unary minus followed by digits.<br>A string argument; see below.                                                                                                                                                                                                                                                                                              |

**Matching Expression**

The `' : '` matching operator shall compare the string resulting from the evaluation of *expr1* with the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax shall be that defined in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions, except that all patterns are anchored to the beginning of the string (that is, only sequences starting at the first character of a string are matched by the regular expression) and, therefore, it is unspecified whether `' ^ '` is a special character in that context. Usually, the matching operator shall return a string representing the number of characters matched (`' 0 '` on failure). Alternatively, if the pattern contains at least one regular expression subexpression `" [ \ ( . . . \ ) ] "`, the string corresponding to `" \1 "` shall be returned.

**String Operand**

A string argument is an argument that cannot be identified as an *integer* argument or as one of the expression operator symbols shown in the OPERANDS section.

The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

**EXIT STATUS**

The following exit values shall be returned:

- 0 The *expression* evaluates to neither null nor zero.
- 1 The *expression* evaluates to null or zero.
- 2 Invalid *expression*.
- >2 An error occurred.

**CONSEQUENCES OF ERRORS**

Default.

**APPLICATION USAGE**

After argument processing by the shell, *expr* is not required to be able to tell the difference between an operator and an operand except by the value. If `"$a"` is `' = '`, the command:

```
expr $a = ' = '
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they all may be taken as the `' = '` operator). The following works reliably:

```
expr X$a = X=
```

Also note that this volume of IEEE Std 1003.1-2001 permits implementations to extend utilities. The *expr* utility permits the integer arguments to be preceded with a unary minus. This means that an integer argument could look like an option. Therefore, the conforming application must employ the `"--"` construct of Guideline 10 of the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines to protect its operands if there is any chance the first operand might be a negative integer (or any string with a leading minus).

**EXAMPLES**

The *expr* utility has a rather difficult syntax:

- Many of the operators are also shell control operators or reserved words, so they have to be escaped on the command line.

- Each part of the expression is composed of separate arguments, so liberal usage of <blank>s is required. For example:

| Invalid                       | Valid                              |
|-------------------------------|------------------------------------|
| <code>expr 1+2</code>         | <code>expr 1 + 2</code>            |
| <code>expr "1 + 2"</code>     | <code>expr 1 + 2</code>            |
| <code>expr 1 + (2 * 3)</code> | <code>expr 1 + \( 2 \* 3 \)</code> |

In many cases, the arithmetic and string features provided as part of the shell command language are easier to use than their equivalents in *expr*. Newly written scripts should avoid *expr* in favor of the new features within the shell; see Section 2.5 (on page 33) and Section 2.6.4 (on page 41).

The following command:

```
a=$(expr $a + 1)
```

adds 1 to the variable *a*.

The following command, for "*\$a*" equal to either **/usr/abc/file** or just **file**:

```
expr $a : '.*\/(.*\)' \| $a
```

returns the last segment of a pathname (that is, **file**). Applications should avoid the character **'/'** used alone as an argument; *expr* may interpret it as the division operator.

The following command:

```
expr "//$a" : '.*\/(.*\)'
```

is a better representation of the previous example. The addition of the **"//"** characters eliminates any ambiguity about the division operator and simplifies the whole expression. Also note that pathnames may contain characters contained in the *IFS* variable and should be quoted to avoid having "*\$a*" expand into multiple arguments.

The following command:

```
expr "$VAR" : '.*'
```

returns the number of characters in *VAR*.

#### RATIONALE

In an early proposal, EREs were used in the matching expression syntax. This was changed to BREs to avoid breaking historical applications.

The use of a leading circumflex in the BRE is unspecified because many historical implementations have treated it as a special character, despite their system documentation. For example:

```
expr foo : ^foo expr ^foo : ^foo
```

return 3 and 0, respectively, on those systems; their documentation would imply the reverse. Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on this undocumented feature.

#### FUTURE DIRECTIONS

None.

16796 **SEE ALSO**

16797           Section 2.5 (on page 33), Section 2.6.4 (on page 41)

16798 **CHANGE HISTORY**

16799           First released in Issue 2.

16800 **Issue 5**

16801           The FUTURE DIRECTIONS section is added.

16802 **Issue 6**

16803           The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE  
16804           PASC Interpretation 1003.2 #104.

16805           The normative text is reworded to avoid use of the term “must” for application requirements.

16806 **NAME**

16807 false — return false value

16808 **SYNOPSIS**

16809 false

16810 **DESCRIPTION**16811 The *false* utility shall return with a non-zero exit code.16812 **OPTIONS**

16813 None.

16814 **OPERANDS**

16815 None.

16816 **STDIN**

16817 Not used.

16818 **INPUT FILES**

16819 None.

16820 **ENVIRONMENT VARIABLES**

16821 None.

16822 **ASYNCHRONOUS EVENTS**

16823 Default.

16824 **STDOUT**

16825 Not used.

16826 **STDERR**

16827 Not used.

16828 **OUTPUT FILES**

16829 None.

16830 **EXTENDED DESCRIPTION**

16831 None.

16832 **EXIT STATUS**16833 The *false* utility shall always exit with a value other than zero.16834 **CONSEQUENCES OF ERRORS**

16835 Default.

16836 **APPLICATION USAGE**

16837 None.

16838 **EXAMPLES**

16839 None.

16840 **RATIONALE**

16841 None.

16842 **FUTURE DIRECTIONS**

16843 None.

16844 **SEE ALSO**16845 *true*

16846 **CHANGE HISTORY**

16847 First released in Issue 2.

16848 **Issue 6**

16849 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/24 is applied, changing the STDERR  
16850 section from “None.” to “Not used.” for alignment with Section 1.11 (on page 20).

16851 **NAME**

16852       fc — process the command history list

16853 **SYNOPSIS**16854 UP       fc [-r] [-e *editor*] [*first* [*last*]]16855       fc -l [-nr] [*first* [*last*]]16856       fc -s [*old=new*] [*first*]

16857

16858 **DESCRIPTION**16859       The *fc* utility shall list, or shall edit and re-execute, commands previously entered to an  
16860       interactive *sh*.

16861       The command history list shall reference commands by number. The first number in the list is  
 16862       selected arbitrarily. The relationship of a number to its command shall not change except when  
 16863       the user logs in and no other process is accessing the list, at which time the system may reset the  
 16864       numbering to start the oldest retained command at another number (usually 1). When the  
 16865       number reaches an implementation-defined upper limit, which shall be no smaller than the  
 16866       value in *HISTSIZE* or 32 767 (whichever is greater), the shell may wrap the numbers, starting the  
 16867       next command with a lower number (usually 1). However, despite this optional wrapping of  
 16868       numbers, *fc* shall maintain the time-ordering sequence of the commands. For example, if four  
 16869       commands in sequence are given the numbers 32 766, 32 767, 1 (wrapped), and 2 as they are  
 16870       executed, command 32 767 is considered the command previous to 1, even though its number is  
 16871       higher.

16872       When commands are edited (when the **-l** option is not specified), the resulting lines shall be  
 16873       entered at the end of the history list and then re-executed by *sh*. The *fc* command that caused the  
 16874       editing shall not be entered into the history list. If the editor returns a non-zero exit status, this  
 16875       shall suppress the entry into the history list and the command re-execution. Any command line  
 16876       variable assignments or redirection operators used with *fc* shall affect both the *fc* command itself  
 16877       as well as the command that results; for example:

16878       fc -s -- -l 2 &gt;/dev/null

16879       reinvokes the previous command, suppressing standard error for both *fc* and the previous  
 16880       command.

16881 **OPTIONS**

16882       The *fc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 16883       Utility Syntax Guidelines.

16884       The following options shall be supported:

16885       **-e editor**     Use the editor named by *editor* to edit the commands. The *editor* string is a utility  
 16886       name, subject to search via the *PATH* variable (see the Base Definitions volume of  
 16887       IEEE Std 1003.1-2001, Chapter 8, Environment Variables). The value in the *FCEDIT*  
 16888       variable shall be used as a default when **-e** is not specified. If *FCEDIT* is null or  
 16889       unset, *ed* shall be used as the editor.

16890       **-l**           (The letter ell.) List the commands rather than invoking an editor on them. The  
 16891       commands shall be written in the sequence indicated by the *first* and *last* operands,  
 16892       as affected by **-r**, with each command preceded by the command number.

16893       **-n**           Suppress command numbers when listing with **-l**.

16894       **-r**           Reverse the order of the commands listed (with **-l**) or edited (with neither **-l** nor  
 16895       **-s**).



16896            **-s**            Re-execute the command without invoking an editor.

## 16897 OPERANDS

16898            The following operands shall be supported:

16899            *first, last*        Select the commands to list or edit. The number of previous commands that can be  
16900                                accessed shall be determined by the value of the *HISTSIZE* variable. The value of  
16901                                *first* or *last* or both shall be one of the following:

16902                                **[+]***number*        A positive number representing a command number; command  
16903                                numbers can be displayed with the **-l** option.

16904                                **-***number*            A negative decimal number representing the command that was  
16905                                executed *number* of commands previously. For example, **-1** is the  
16906                                immediately previous command.

16907                                *string*            A string indicating the most recently entered command that begins  
16908                                with that string. If the *old=new* operand is not also specified with **-s**,  
16909                                the string form of the *first* operand cannot contain an embedded  
16910                                equal sign.

16911            When the synopsis form with **-s** is used:

- 16912                                • If *first* is omitted, the previous command shall be used.

16913            For the synopsis forms without **-s**:

- 16914                                • If *last* is omitted, *last* shall default to the previous command when **-l** is  
16915                                specified; otherwise, it shall default to *first*.

- 16916                                • If *first* and *last* are both omitted, the previous 16 commands shall be listed or  
16917                                the previous single command shall be edited (based on the **-l** option).

- 16918                                • If *first* and *last* are both present, all of the commands from *first* to *last* shall be  
16919                                edited (without **-l**) or listed (with **-l**). Editing multiple commands shall be  
16920                                accomplished by presenting to the editor all of the commands at one time, each  
16921                                command starting on a new line. If *first* represents a newer command than *last*,  
16922                                the commands shall be listed or edited in reverse sequence, equivalent to using  
16923                                **-r**. For example, the following commands on the first line are equivalent to the  
16924                                corresponding commands on the second:

```
16925 fc -r 10 20 fc 30 40
16926 fc 20 10 fc -r 40 30
```

- 16927                                • When a range of commands is used, it shall not be an error to specify *first* or *last*  
16928                                values that are not in the history list; *fc* shall substitute the value representing  
16929                                the oldest or newest command in the list, as appropriate. For example, if there  
16930                                are only ten commands in the history list, numbered 1 to 10:

```
16931 fc -l
16932 fc 1 99
```

16933                                shall list and edit, respectively, all ten commands.

16934            *old=new*            Replace the first occurrence of string *old* in the commands to be re-executed by the  
16935                                string *new*.

16936 **STDIN**

16937 Not used.

16938 **INPUT FILES**

16939 None.

16940 **ENVIRONMENT VARIABLES**16941 The following environment variables shall affect the execution of *fc*:

16942 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for  
 16943 the **-e** *editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be  
 16944 used as the editor.

16945 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is  
 16946 not set, the shell may attempt to access or create a file **.sh\_history** in the directory  
 16947 referred to by the *HOME* environment variable. If the shell cannot obtain both read  
 16948 and write access to, or create, the history file, it shall use an unspecified  
 16949 mechanism that allows the history to operate properly. (References to history  
 16950 “file” in this section shall be understood to mean this unspecified mechanism in  
 16951 such cases.) An implementation may choose to access this variable only when  
 16952 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt  
 16953 to retrieve entries from, or add entries to, the file, as the result of commands issued  
 16954 by the user, the file named by the *ENV* variable, or implementation-defined system  
 16955 start-up files. In some historical shells, the history file is initialized just after the  
 16956 *ENV* file has been processed. Therefore, it is implementation-defined whether  
 16957 changes made to *HISTFILE* after the history file has been initialized are effective.  
 16958 Implementations may choose to disable the history list mechanism for users with  
 16959 appropriate privileges who do not set *HISTFILE*; the specific circumstances under  
 16960 which this occurs are implementation-defined. If more than one instance of the  
 16961 shell is using the same history file, it is unspecified how updates to the history file  
 16962 from those shells interact. As entries are deleted from the history file, they shall be  
 16963 deleted oldest first. It is unspecified when history file entries are physically  
 16964 removed from the history file.

16965 *HISTSIZE* Determine a decimal number representing the limit to the number of previous  
 16966 commands that are accessible. If this variable is unset, an unspecified default  
 16967 greater than or equal to 128 shall be used. The maximum number of commands in  
 16968 the history list is unspecified, but shall be at least 128. An implementation may  
 16969 choose to access this variable only when initializing the history file, as described  
 16970 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*  
 16971 after the history file has been initialized are effective.

16972 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 16973 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 16974 Internationalization Variables for the precedence of internationalization variables  
 16975 used to determine the values of locale categories.)

16976 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 16977 internationalization variables.

16978 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 16979 characters (for example, single-byte as opposed to multi-byte characters in  
 16980 arguments and input files).

16981 *LC\_MESSAGES*

16982 Determine the locale that should be used to affect the format and contents of  
 16983 diagnostic messages written to standard error.

16984 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 16985 ASYNCHRONOUS EVENTS

16986 Default.

## 16987 STDOUT

16988 When the **-l** option is used to list commands, the format of each command in the list shall be as follows:

16990 "%d\t%s\n", <line number>, <command>

16991 If both the **-l** and **-n** options are specified, the format of each command shall be:

16992 "\t%s\n", <command>

16993 If the <command> consists of more than one line, the lines after the first shall be displayed as:

16994 "\t%s\n", <continued-command>

## 16995 STDERR

16996 The standard error shall be used only for diagnostic messages.

## 16997 OUTPUT FILES

16998 None.

## 16999 EXTENDED DESCRIPTION

17000 None.

## 17001 EXIT STATUS

17002 The following exit values shall be returned:

17003 0 Successful completion of the listing.

17004 >0 An error occurred.

17005 Otherwise, the exit status shall be that of the commands executed by *fc*.

## 17006 CONSEQUENCES OF ERRORS

17007 Default.

## 17008 APPLICATION USAGE

17009 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file descriptors as part of the *fc* command can produce unexpected results. For example, if *vi* is the *FCEDIT* editor, the command:

17012 *fc -s | more*

17013 does not work correctly on many systems.

17014 Users on windowing systems may want to have separate history files for each window by setting *HISTFILE* as follows:

17016 *HISTFILE=\$HOME/.sh\_hist\$\$*

## 17017 EXAMPLES

17018 None.

## 17019 RATIONALE

17020 This utility is based on the *fc* built-in of the KornShell.

17021 An early proposal specified the **-e** option as [**-e editor [old= new ]**], which is not historical practice. Historical practice in *fc* of either [**-e editor**] or [**-e - [old= new ]**] is acceptable, but not both together. To clarify this, a new option **-s** was introduced replacing the [**-e -**]. This resolves the conflict and makes *fc* conform to the Utility Syntax Guidelines.

17025 **HISTFILE** Some implementations of the KornShell check for the superuser and do not create  
 17026 a history file unless *HISTFILE* is set. This is done primarily to avoid creating  
 17027 unlinked files in the root file system when logging in during single-user mode.  
 17028 *HISTFILE* must be set for the superuser to have history.

17029 **HISTSIZE** Needed to limit the size of history files. It is the intent of the standard developers  
 17030 that when two shells share the same history file, commands that are entered in one  
 17031 shell shall be accessible by the other shell. Because of the difficulties of  
 17032 synchronization over a network, the exact nature of the interaction is unspecified.

17033 The initialization process for the history file can be dependent on the system start-up files, in  
 17034 that they may contain commands that effectively preempt the settings the user has for *HISTFILE*  
 17035 and *HISTSIZE*. For example, function definition commands are recorded in the history file. If the  
 17036 system administrator includes function definitions in some system start-up file called before the  
 17037 *ENV* file, the history file is initialized before the user can influence its characteristics. In some  
 17038 historical shells, the history file is initialized just after the *ENV* file has been processed. Because  
 17039 of these situations, the text requires the initialization process to be implementation-defined.

17040 Consideration was given to omitting the *fc* utility in favor of the command line editing feature in  
 17041 *sh*. For example, in *vi* editing mode, typing "<ESC> v" is equivalent to:

17042 EDITOR=vi fc

17043 However, the *fc* utility allows the user the flexibility to edit multiple commands simultaneously  
 17044 (such as *fc* 10 20) and to use editors other than those supported by *sh* for command line editing.

17045 In the KornShell, the alias **r** ("re-do") is preset to *fc -e* (equivalent to the POSIX *fc -s*). This is  
 17046 probably an easier command name to remember than *fc* ("fix command"), but it does not meet  
 17047 the Utility Syntax Guidelines. Renaming *fc* to *hist* or *redo* was considered, but since this  
 17048 description closely matches historical KornShell practice already, such a renaming was seen as  
 17049 gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*,  
 17050 *grep*, or *yacc* are standardized by POSIX.

17051 Command numbers have no ordering effects; they are like serial numbers. The **-r** option and  
 17052 **-number** operand address the sequence of command execution, regardless of serial numbers. So,  
 17053 for example, if the command number wrapped back to 1 at some arbitrary point, there would be  
 17054 no ambiguity associated with traversing the wrap point. For example, if the command history  
 17055 were:

17056 32766: echo 1  
 17057 32767: echo 2  
 17058 1: echo 3

17059 the number **-2** refers to command 32 767 because it is the second previous command, regardless  
 17060 of serial number.

17061 **FUTURE DIRECTIONS**

17062 None.

17063 **SEE ALSO**

17064 *sh*

17065 **CHANGE HISTORY**

17066 First released in Issue 4.

17067 **Issue 5**

17068 The FUTURE DIRECTIONS section is added.

17069 **Issue 6**

17070 This utility is marked as part of the User Portability Utilities option.

17071 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to  
17072 “directory referred to by the *HOME* environment variable”.

17073 **NAME**

17074           fg — run jobs in the foreground

17075 **SYNOPSIS**17076 UP       fg [*job\_id*]

17077

17078 **DESCRIPTION**

17079           If job control is enabled (see the description of *set -m*), the *fg* utility shall move a background job  
 17080           from the current environment (see Section 2.12 (on page 61)) into the foreground.

17081           Using *fg* to place a job into the foreground shall remove its process ID from the list of those  
 17082           “known in the current shell execution environment”; see Section 2.9.3.1 (on page 50).

17083 **OPTIONS**

17084           None.

17085 **OPERANDS**

17086           The following operand shall be supported:

17087           *job\_id*       Specify the job to be run as a foreground job. If no *job\_id* operand is given, the  
 17088                           *job\_id* for the job that was most recently suspended, placed in the background, or  
 17089                           run as a background job shall be used. The format of *job\_id* is described in the Base  
 17090                           Definitions volume of IEEE Std 1003.1-2001, Section 3.203, Job Control Job ID.

17091 **STDIN**

17092           Not used.

17093 **INPUT FILES**

17094           None.

17095 **ENVIRONMENT VARIABLES**17096           The following environment variables shall affect the execution of *fg*:

17097           *LANG*        Provide a default value for the internationalization variables that are unset or null.  
 17098                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 17099                           Internationalization Variables for the precedence of internationalization variables  
 17100                           used to determine the values of locale categories.)

17101           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 17102                           internationalization variables.

17103           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 17104                           characters (for example, single-byte as opposed to multi-byte characters in  
 17105                           arguments).

17106           *LC\_MESSAGES*

17107                           Determine the locale that should be used to affect the format and contents of  
 17108                           diagnostic messages written to standard error.

17109 XSI       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.17110 **ASYNCHRONOUS EVENTS**

17111           Default.

17112 **STDOUT**17113           The *fg* utility shall write the command line of the job to standard output in the following format:17114           "%s\n", <*command*>

**17115 STDERR**

17116 The standard error shall be used only for diagnostic messages.

**17117 OUTPUT FILES**

17118 None.

**17119 EXTENDED DESCRIPTION**

17120 None.

**17121 EXIT STATUS**

17122 The following exit values shall be returned:

17123 0 Successful completion.

17124 >0 An error occurred.

**17125 CONSEQUENCES OF ERRORS**

17126 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the foreground.

**17128 APPLICATION USAGE**

17129 The *fg* utility does not work as expected when it is operating in its own utility execution environment because that environment has no applicable jobs to manipulate. See the APPLICATION USAGE section for *bg*. For this reason, *fg* is generally implemented as a shell regular built-in.

**17133 EXAMPLES**

17134 None.

**17135 RATIONALE**

17136 The extensions to the shell specified in this volume of IEEE Std 1003.1-2001 have mostly been based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also based on the KornShell. The standard developers examined the characteristics of the C shell versions of these utilities and found that differences exist. Despite widespread use of the C shell, the KornShell versions were selected for this volume of IEEE Std 1003.1-2001 to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command line editing features).

**17143 FUTURE DIRECTIONS**

17144 None.

**17145 SEE ALSO**

17146 Section 2.9.3.1 (on page 50), Section 2.12 (on page 61), *bg*, *kill*, *jobs*, *wait*

**17147 CHANGE HISTORY**

17148 First released in Issue 4.

**17149 Issue 6**

17150 This utility is marked as part of the User Portability Utilities option.

17151 The APPLICATION USAGE section is added.

17152 The JC marking is removed from the SYNOPSIS since job control is mandatory in this issue.

17153 **NAME**

17154 file — determine file type

17155 **SYNOPSIS**17156 UP file [-dh] [-M *file*] [-m *file*] *file* ...17157 file -i [-h] *file* ...

17158

17159 **DESCRIPTION**

17160 The *file* utility shall perform a series of tests in sequence on each specified *file* in an attempt to  
 17161 classify it:

- 17162 1. If *file* does not exist, cannot be read, or its file status could not be determined, the output  
 17163 shall indicate that the file was processed, but that its type could not be determined.
- 17164 2. If the file is not a regular file, its file type shall be identified. The file types directory, FIFO,  
 17165 socket, block special, and character special shall be identified as such. Other  
 17166 implementation-defined file types may also be identified. If *file* is a symbolic link, by  
 17167 default the link shall be resolved and *file* shall test the type of file referenced by the  
 17168 symbolic link. (See the **-h** and **-i** options below.)
- 17169 3. If the length of *file* is zero, it shall be identified as an empty file.
- 17170 4. The *file* utility shall examine an initial segment of *file* and shall make a guess at identifying  
 17171 its contents based on position-sensitive tests. (The answer is not guaranteed to be correct;  
 17172 see the **-d**, **-M**, and **-m** options below.)
- 17173 5. The *file* utility shall examine *file* and make a guess at identifying its contents based on  
 17174 context-sensitive default system tests. (The answer is not guaranteed to be correct.)
- 17175 6. The file shall be identified as a data file.

17176 If *file* does not exist, cannot be read, or its file status could not be determined, the output shall  
 17177 indicate that the file was processed, but that its type could not be determined.

17178 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file  
 17179 referenced by the symbolic link.

17180 **OPTIONS**

17181 The *file* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 17182 Utility Syntax Guidelines, except that the order of the **-m**, **-d**, and **-M** options shall be  
 17183 significant.

17184 The following options shall be supported by the implementation:

- |       |                       |                                                                                                       |
|-------|-----------------------|-------------------------------------------------------------------------------------------------------|
| 17185 | <b>-d</b>             | Apply any position-sensitive default system tests and context-sensitive default                       |
| 17186 |                       | system tests to the file. This is the default if no <b>-M</b> or <b>-m</b> option is specified.       |
| 17187 | <b>-h</b>             | When a symbolic link is encountered, identify the file as a symbolic link. If <b>-h</b> is            |
| 17188 |                       | not specified and <i>file</i> is a symbolic link that refers to a nonexistent file, <i>file</i> shall |
| 17189 |                       | identify the file as a symbolic link, as if <b>-h</b> had been specified.                             |
| 17190 | <b>-i</b>             | If a file is a regular file, do not attempt to classify the type of the file further, but             |
| 17191 |                       | identify the file as specified in the STDOUT section.                                                 |
| 17192 | <b>-M <i>file</i></b> | Specify the name of a file containing position-sensitive tests that shall be applied to               |
| 17193 |                       | a file in order to classify it (see the EXTENDED DESCRIPTION). No position-                           |
| 17194 |                       | sensitive default system tests nor context-sensitive default system tests shall be                    |
| 17195 |                       | applied unless the <b>-d</b> option is also specified.                                                |



17196        **-m file**        Specify the name of a file containing position-sensitive tests that shall be applied to  
 17197                                a file in order to classify it (see the EXTENDED DESCRIPTION).

17198        If the **-m** option is specified without specifying the **-d** option or the **-M** option, position-  
 17199        sensitive default system tests shall be applied after the position-sensitive tests specified by the  
 17200        **-m** option. If the **-M** option is specified with the **-d** option, the **-m** option, or both, or the **-m**  
 17201        option is specified with the **-d** option, the concatenation of the position-sensitive tests specified  
 17202        by these options shall be applied in the order specified by the appearance of these options. If a  
 17203        **-M** or **-m file** option-argument is **-**, the results are unspecified.

#### 17204 OPERANDS

17205        The following operand shall be supported:

17206        **file**                A pathname of a file to be tested.

#### 17207 STDIN

17208        Not used.

#### 17209 INPUT FILES

17210        The *file* can be any file type.

#### 17211 ENVIRONMENT VARIABLES

17212        The following environment variables shall affect the execution of *file*:

17213        **LANG**                Provide a default value for the internationalization variables that are unset or null.  
 17214                                (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 17215                                Internationalization Variables for the precedence of internationalization variables  
 17216                                used to determine the values of locale categories.)

17217        **LC\_ALL**              If set to a non-empty string value, override the values of all the other  
 17218                                internationalization variables.

17219        **LC\_CTYPE**            Determine the locale for the interpretation of sequences of bytes of text data as  
 17220                                characters (for example, single-byte as opposed to multi-byte characters in  
 17221                                arguments and input files).

17222        **LC\_MESSAGES**

17223                                Determine the locale that should be used to affect the format and contents of  
 17224                                diagnostic messages written to standard error and informative messages written to  
 17225                                standard output.

17226 xsi        **NLSPATH**            Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

#### 17227 ASYNCHRONOUS EVENTS

17228        Default.

#### 17229 STDOUT

17230        In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

17231        "%s: %s\n", <file>, <type>

17232        The values for <type> are unspecified, except that in the POSIX locale, if *file* is identified as one  
 17233        of the types listed in the following table, <type> shall contain (but is not limited to) the  
 17234        corresponding string, unless the file is identified by a position-sensitive test specified by a **-M** or  
 17235        **-m** option. Each space shown in the strings shall be exactly one <space>.

Table 4-8 File Utility Output Strings

| If <i>file</i> is:                                           | <type> shall contain the string: | Notes |
|--------------------------------------------------------------|----------------------------------|-------|
| Nonexistent                                                  | cannot open                      |       |
| Block special                                                | block special                    | 1     |
| Character special                                            | character special                | 1     |
| Directory                                                    | directory                        | 1     |
| FIFO                                                         | fifo                             | 1     |
| Socket                                                       | socket                           | 1     |
| Symbolic link                                                | symbolic link to                 | 1     |
| Regular file                                                 | regular file                     | 1,2   |
| Empty regular file                                           | empty                            | 3     |
| Regular file that cannot be read                             | cannot open                      | 3     |
| Executable binary                                            | executable                       | 4,6   |
| <i>ar</i> archive library (see <i>ar</i> )                   | archive                          | 4,6   |
| Extended <i>cpio</i> format (see <i>pax</i> )                | cpio archive                     | 4,6   |
| Extended <i>tar</i> format (see <b>ustar</b> in <i>pax</i> ) | tar archive                      | 4,6   |
| Shell script                                                 | commands text                    | 5,6   |
| C-language source                                            | c program text                   | 5,6   |
| FORTRAN source                                               | fortran program text             | 5,6   |
| Regular file whose type cannot be determined                 | data                             |       |

**Notes:**

1. This is a file type test.
2. This test is applied only if the **-i** option is specified.
3. This test is applied only if the **-i** option is not specified.
4. This is a position-sensitive default system test.
5. This is a context-sensitive default system test.
6. Position-sensitive default system tests and context-sensitive default system tests are not applied if the **-M** option is specified unless the **-d** option is also specified.

In the POSIX locale, if *file* is identified as a symbolic link (see the **-h** option), the following alternative output format shall be used:

"%s: %s %s\n", <file>, <type>, <contents of link>"

If the file named by the *file* operand does not exist, cannot be read, or the type of the file named by the *file* operand cannot be determined, this shall not be considered an error that affects the exit status.

**STDERR**

The standard error shall be used only for diagnostic messages.

**OUTPUT FILES**

None.

**EXTENDED DESCRIPTION**

A file specified as an option-argument to the **-m** or **-M** options shall contain one position-sensitive test per line, which shall be applied to the file. If the test succeeds, the message field of the line shall be printed and no further tests shall be applied, with the exception that tests on immediately following lines beginning with a single ' > ' character shall be applied.

|       |               |                                                                                                                     |
|-------|---------------|---------------------------------------------------------------------------------------------------------------------|
| 17280 |               | Each line shall be composed of the following four <blank>-separated fields:                                         |
| 17281 | <i>offset</i> | An unsigned number (optionally preceded by a single ' <i>&gt;</i> ' character) specifying                           |
| 17282 |               | the <i>offset</i> , in bytes, of the value in the file that is to be compared against the <i>value</i>              |
| 17283 |               | field of the line. If the file is shorter than the specified offset, the test shall fail.                           |
| 17284 |               | If the <i>offset</i> begins with the character ' <i>&gt;</i> ', the test contained in the line shall not be         |
| 17285 |               | applied to the file unless the test on the last line for which the <i>offset</i> did not begin                      |
| 17286 |               | with a ' <i>&gt;</i> ' was successful. By default, the <i>offset</i> shall be interpreted as an unsigned            |
| 17287 |               | decimal number. With a leading 0x or 0X, the <i>offset</i> shall be interpreted as a                                |
| 17288 |               | hexadecimal number; otherwise, with a leading 0, the <i>offset</i> shall be interpreted as                          |
| 17289 |               | an octal number.                                                                                                    |
| 17290 | <i>type</i>   | The type of the value in the file to be tested. The type shall consist of the type                                  |
| 17291 |               | specification characters c, d, f, s, and u, specifying character, signed decimal,                                   |
| 17292 |               | floating point, string, and unsigned decimal, respectively.                                                         |
| 17293 |               | The <i>type</i> string shall be interpreted as the bytes from the file starting at the                              |
| 17294 |               | specified <i>offset</i> and including the same number of bytes specified by the <i>value</i> field.                 |
| 17295 |               | If insufficient bytes remain in the file past the <i>offset</i> to match the <i>value</i> field, the test           |
| 17296 |               | shall fail.                                                                                                         |
| 17297 |               | The type specification characters d, f, and u can be followed by an optional                                        |
| 17298 |               | unsigned decimal integer that specifies the number of bytes represented by the                                      |
| 17299 |               | type. The type specification character f can be followed by an optional F, D, or L,                                 |
| 17300 |               | indicating that the value is of type <b>float</b> , <b>double</b> , or <b>long double</b> , respectively. The       |
| 17301 |               | type specification characters d and u can be followed by an optional C, S, I, or L,                                 |
| 17302 |               | indicating that the value is of type <b>char</b> , <b>short</b> , <b>int</b> , or <b>long</b> , respectively.       |
| 17303 |               | The default number of bytes represented by the type specifiers d, f, and u shall                                    |
| 17304 |               | correspond to their respective C-language types as follows. If the system claims                                    |
| 17305 |               | conformance to the C-Language Development Utilities option, those specifiers                                        |
| 17306 |               | shall correspond to the default sizes used in the c99 utility. Otherwise, the default                               |
| 17307 |               | sizes shall be implementation-defined.                                                                              |
| 17308 |               | For the type specifier characters d and u, the default number of bytes shall                                        |
| 17309 |               | correspond to the size of a basic integer type of the implementation. For these                                     |
| 17310 |               | specifier characters, the implementation shall support values of the optional                                       |
| 17311 |               | number of bytes to be converted corresponding to the number of bytes in the C-                                      |
| 17312 |               | language types <b>char</b> , <b>short</b> , <b>int</b> , or <b>long</b> . These numbers can also be specified by an |
| 17313 |               | application as the characters C, S, I, and L, respectively. The byte order used when                                |
| 17314 |               | interpreting numeric values is implementation-defined, but shall correspond to the                                  |
| 17315 |               | order in which a constant of the corresponding type is stored in memory on the                                      |
| 17316 |               | system.                                                                                                             |
| 17317 |               | For the type specifier f, the default number of bytes shall correspond to the                                       |
| 17318 |               | number of bytes in the basic double precision floating-point data type of the                                       |
| 17319 |               | underlying implementation. The implementation shall support values of the                                           |
| 17320 |               | optional number of bytes to be converted corresponding to the number of bytes in                                    |
| 17321 |               | the C-language types <b>float</b> , <b>double</b> , and <b>long double</b> . These numbers can also be              |
| 17322 |               | specified by an application as the characters F, D, and L, respectively.                                            |
| 17323 |               | All type specifiers, except for s, can be followed by a mask specifier of the form                                  |
| 17324 |               | &number. The mask value shall be AND'ed with the value of the input file before                                     |
| 17325 |               | the comparison with the <i>value</i> field of the line is made. By default, the mask shall                          |
| 17326 |               | be interpreted as an unsigned decimal number. With a leading 0x or 0X, the mask                                     |
| 17327 |               | shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading                                   |

|       |                    |                                                                                                                  |
|-------|--------------------|------------------------------------------------------------------------------------------------------------------|
| 17328 |                    | 0, the mask shall be interpreted as an unsigned octal number.                                                    |
| 17329 |                    | The strings <b>byte</b> , <b>short</b> , <b>long</b> , and <b>string</b> shall also be supported as type fields, |
| 17330 |                    | being interpreted as dC, dS, dL, and s, respectively.                                                            |
| 17331 | <i>value</i>       | The <i>value</i> to be compared with the value from the file.                                                    |
| 17332 |                    | If the specifier from the type field is s or <b>string</b> , then interpret the value as a string.               |
| 17333 |                    | Otherwise, interpret it as a number. If the value is a string, then the test shall                               |
| 17334 |                    | succeed only when a string value exactly matches the bytes from the file.                                        |
| 17335 |                    | If the <i>value</i> is a string, it can contain the following sequences:                                         |
| 17336 | <i>\character</i>  | The backslash-escape sequences as specified in the Base                                                          |
| 17337 |                    | Definitions volume of IEEE Std 1003.1-2001, Table 5-1, Escape                                                    |
| 17338 |                    | Sequences and Associated Actions ('\\', '\a', '\b', '\f',                                                        |
| 17339 |                    | '\n', '\r', '\t', '\v'). The results of using any other                                                          |
| 17340 |                    | character, other than an octal digit, following the backslash are                                                |
| 17341 |                    | unspecified.                                                                                                     |
| 17342 | <i>\octal</i>      | Octal sequences that can be used to represent characters with                                                    |
| 17343 |                    | specific coded values. An octal sequence shall consist of a                                                      |
| 17344 |                    | backslash followed by the longest sequence of one, two, or three                                                 |
| 17345 |                    | octal-digit characters (01234567). If the size of a byte on the                                                  |
| 17346 |                    | system is greater than 9 bits, the valid escape sequence used to                                                 |
| 17347 |                    | represent a byte is implementation-defined.                                                                      |
| 17348 |                    | By default, any value that is not a string shall be interpreted as a signed decimal                              |
| 17349 |                    | number. Any such value, with a leading 0x or 0X, shall be interpreted as an                                      |
| 17350 |                    | unsigned hexadecimal number; otherwise, with a leading zero, the value shall be                                  |
| 17351 |                    | interpreted as an unsigned octal number.                                                                         |
| 17352 |                    | If the value is not a string, it can be preceded by a character indicating the                                   |
| 17353 |                    | comparison to be performed. Permissible characters and the comparisons they                                      |
| 17354 |                    | specify are as follows:                                                                                          |
| 17355 | =                  | The test shall succeed if the value from the file equals the <i>value</i> field.                                 |
| 17356 | <                  | The test shall succeed if the value from the file is less than the <i>value</i> field.                           |
| 17357 | >                  | The test shall succeed if the value from the file is greater than the <i>value</i> field.                        |
| 17358 | &                  | The test shall succeed if all of the set bits in the <i>value</i> field are set in the value                     |
| 17359 |                    | from the file.                                                                                                   |
| 17360 | ^                  | The test shall succeed if at least one of the set bits in the <i>value</i> field is not set in                   |
| 17361 |                    | the value from the file.                                                                                         |
| 17362 | x                  | The test shall succeed if the file is large enough to contain a value of the type                                |
| 17363 |                    | specified starting at the offset specified.                                                                      |
| 17364 | <i>message</i>     | The <i>message</i> to be printed if the test succeeds. The <i>message</i> shall be interpreted                   |
| 17365 |                    | using the notation for the <i>printf</i> formatting specification; see <i>printf</i> . If the <i>value</i>       |
| 17366 |                    | field was a string, then the value from the file shall be the argument for the <i>printf</i>                     |
| 17367 |                    | formatting specification; otherwise, the value from the file shall be the argument.                              |
| 17368 | <b>EXIT STATUS</b> |                                                                                                                  |
| 17369 |                    | The following exit values shall be returned:                                                                     |
| 17370 | 0                  | Successful completion.                                                                                           |

17371 >0 An error occurred.

## 17372 CONSEQUENCES OF ERRORS

17373 Default.

## 17374 APPLICATION USAGE

17375 The *file* utility can only be required to guess at many of the file types because only exhaustive  
17376 testing can determine some types with certainty. For example, binary data on some  
17377 implementations might match the initial segment of an executable or a *tar* archive.

17378 Note that the table indicates that the output contains the stated string. Systems may add text  
17379 before or after the string. For executables, as an example, the machine architecture and various  
17380 facts about how the file was link-edited may be included. Note also that on systems that  
17381 recognize shell script files starting with "#!" as executable files, these may be identified as  
17382 executable binary files rather than as shell scripts.

## 17383 EXAMPLES

17384 Determine whether an argument is a binary executable file:

```
17385 file "$1" | grep -Fq executable &&
17386 printf "%s is executable.\n" "$1"
```

## 17387 RATIONALE

17388 The **-f** option was omitted because the same effect can (and should) be obtained using the *xargs*  
17389 utility.

17390 Historical versions of the *file* utility attempt to identify the following types of files: symbolic link,  
17391 directory, character special, block special, socket, *tar* archive, *cpio* archive, SCCS archive, archive  
17392 library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler  
17393 source, *nroff*/*troff*/*eqn*/*tbl* source *troff* output, shell script, C shell script, English text, ASCII text,  
17394 various executables, APL workspace, compiled terminfo entries, and CURSES screen images.  
17395 Only those types that are reasonably well specified in POSIX or are directly related to POSIX  
17396 utilities are listed in the table.

17397 Historical systems have used a “magic file” named **/etc/magic** to help identify file types. Because  
17398 it is generally useful for users and scripts to be able to identify special file types, the **-m** flag and  
17399 a portable format for user-created magic files has been specified. No requirement is made that an  
17400 implementation of *file* use this method of identifying files, only that users be permitted to add  
17401 their own classifying tests.

17402 In addition, three options have been added to historical practice. The **-d** flag has been added to  
17403 permit users to cause their tests to follow any default system tests. The **-i** flag has been added to  
17404 permit users to test portably for regular files in shell scripts. The **-M** flag has been added to  
17405 permit users to ignore any default system tests.

17406 The IEEE Std 1003.1-2001 description of default system tests and the interaction between the **-d**,  
17407 **-M**, and **-m** options did not clearly indicate that there were two types of “default system tests”.  
17408 The “position-sensitive tests” determine file types by looking for certain string or binary values  
17409 at specific offsets in the file being examined. These position-sensitive tests were implemented in  
17410 historical systems using the magic file described above. Some of these tests are now built into  
17411 the *file* utility itself on some implementations so the output can provide more detail than can be  
17412 provided by magic files. For example, a magic file can easily identify a **core** file on most  
17413 implementations, but cannot name the program file that dropped the core. A magic file could  
17414 produce output such as:

```
17415 /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1
```

17416 but by building the test into the *file* utility, you could get output such as: |

17417 /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1, from 'testprog'|

17418 These extended built-in tests are still to be treated as position-sensitive default system tests even |

17419 if they are not listed in **/etc/magic** or any other magic file. |

17420 The context-sensitive default system tests were always built into the *file* utility. These tests |

17421 looked for language constructs in text files trying to identify shell scripts, C, FORTRAN, and |

17422 other computer language source files, and even plain text files. With the addition of the **-m** and |

17423 **-M** options the distinction between position-sensitive and context-sensitive default system tests |

17424 became important because the order of testing is important. The context-sensitive system |

17425 default tests should never be applied before any position-sensitive tests even if the **-d** option is |

17426 specified before a **-m** option or **-M** option due to the high probability that the context-sensitive |

17427 system default tests will incorrectly identify arbitrary text files as text files before position- |

17428 sensitive tests specified by the **-m** or **-M** option would be applied to give a more accurate |

17429 identification. |

17430 Leaving the meaning of **-M** – and **-m** – unspecified allows an existing prototype of these |

17431 options to continue to work in a backwards-compatible manner. (In that implementation, **-M** – |

17432 was roughly equivalent to **-d** in IEEE Std 1003.1-2001.) |

17433 The historical **-c** option was omitted as not particularly useful to users or portable shell scripts. |

17434 In addition, a reasonable implementation of the *file* utility would report any errors found each |

17435 time the magic file is read. |

17436 The historical format of the magic file was the same as that specified by the Rationale in the |

17437 ISO POSIX-2:1993 standard for the *offset*, *value*, and *message* fields; however, it used less precise |

17438 type fields than the format specified by the current normative text. The new type field values are |

17439 a superset of the historical ones. |

17440 The following is an example magic file: |

|       |    |           |                     |                                    |
|-------|----|-----------|---------------------|------------------------------------|
| 17441 | 0  | short     | 070707              | cpio archive                       |
| 17442 | 0  | short     | 0143561             | Byte-swapped cpio archive          |
| 17443 | 0  | string    | 070707              | ASCII cpio archive                 |
| 17444 | 0  | long      | 0177555             | Very old archive                   |
| 17445 | 0  | short     | 0177545             | Old archive                        |
| 17446 | 0  | short     | 017437              | Old packed data                    |
| 17447 | 0  | string    | \037\036            | Packed data                        |
| 17448 | 0  | string    | \377\037            | Compacted data                     |
| 17449 | 0  | string    | \037\235            | Compressed data                    |
| 17450 | >2 | byte&0x80 | >0                  | Block compressed                   |
| 17451 | >2 | byte&0x1f | x                   | %d bits                            |
| 17452 | 0  | string    | \032\001            | Compiled Terminfo Entry            |
| 17453 | 0  | short     | 0433                | Curses screen image                |
| 17454 | 0  | short     | 0434                | Curses screen image                |
| 17455 | 0  | string    | <ar>                | System V Release 1 archive         |
| 17456 | 0  | string    | !<arch>\n___.SYMDEF | Archive random library             |
| 17457 | 0  | string    | !<arch>             | Archive                            |
| 17458 | 0  | string    | ARF_BEGARF          | PHIGS clear text archive           |
| 17459 | 0  | long      | 0x137A2950          | Scalable OpenFont binary           |
| 17460 | 0  | long      | 0x137A2951          | Encrypted scalable OpenFont binary |

17461 The use of a basic integer data type is intended to allow the implementation to choose a word |

17462 size commonly used by applications on that architecture. |

17463 **FUTURE DIRECTIONS**

17464       None.

17465 **SEE ALSO**17466       *ar, ls, pax*17467 **CHANGE HISTORY**

17468       First released in Issue 4.

17469 **Issue 6**

17470       This utility is marked as part of the User Portability Utilities option.

17471       Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft  
17472       standard.

17473       IEEE PASC Interpretations 1003.2 #192 and #178 are applied.

17474       IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/25 is applied, making major changes to  
17475       address ambiguities raised in defect reports.17476       IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/26 is applied, making it clear in the  
17477       OPTIONS section that the **-m**, **-d**, and **-M** options do not comply with Guideline 11 of the  
17478       Utility Syntax Guidelines.

17479 **NAME**

17480 find — find files

17481 **SYNOPSIS**17482 find [-H | -L] *path* ... [*operand\_expression* ...]17483 **DESCRIPTION**

17484 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,  
 17485 evaluating a Boolean expression composed of the primaries described in the OPERANDS section  
 17486 for each file encountered.

17487 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail  
 17488 due to path length limitations (unless a *path* operand specified by the application exceeds  
 17489 {PATH\_MAX} requirements).

17490 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an  
 17491 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a  
 17492 diagnostic message to standard error and shall either recover its position in the hierarchy or  
 17493 terminate.

17494 **OPTIONS**

17495 The *find* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 17496 12.2, Utility Syntax Guidelines.

17497 The following options shall be supported by the implementation:

17498 **-H** Cause the file information and file type evaluated for each symbolic link  
 17499 encountered on the command line to be those of the file referenced by the link, and  
 17500 not the link itself. If the referenced file does not exist, the file information and type  
 17501 shall be for the link itself. File information for all symbolic links not on the  
 17502 command line shall be that of the link itself.

17503 **-L** Cause the file information and file type evaluated for each symbolic link to be  
 17504 those of the file referenced by the link, and not the link itself.

17505 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
 17506 an error. The last option specified shall determine the behavior of the utility.

17507 **OPERANDS**

17508 The following operands shall be supported:

17509 The *path* operand is a pathname of a starting point in the directory hierarchy.

17510 The first argument that starts with a '-', or is a '!' or a '(', and all subsequent arguments  
 17511 shall be interpreted as an *expression* made up of the following primaries and operators. In the  
 17512 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal  
 17513 integer optionally preceded by a plus ('+') or minus ('-') sign, as follows:

17514 **+n** More than *n*.

17515 **n** Exactly *n*.

17516 **-n** Less than *n*.

17517 The following primaries shall be supported:

17518 **-name** *pattern*

17519 The primary shall evaluate as true if the basename of the filename being examined  
 17520 matches *pattern* using the pattern matching notation described in Section 2.13 (on  
 17521 page 62).



|       |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 17522 | <b>-nouser</b>       | The primary shall evaluate as true if the file belongs to a user ID for which the <i>getpwuid()</i> function defined in the System Interfaces volume of IEEE Std 1003.1-2001 (or equivalent) returns NULL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 17523 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17524 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17525 | <b>-nogroup</b>      | The primary shall evaluate as true if the file belongs to a group ID for which the <i>getgrgid()</i> function defined in the System Interfaces volume of IEEE Std 1003.1-2001 (or equivalent) returns NULL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 17526 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17527 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17528 | <b>-xdev</b>         | The primary shall always evaluate as true; it shall cause <i>find</i> not to continue descending past directories that have a different device ID ( <i>st_dev</i> , see the <i>stat()</i> function defined in the System Interfaces volume of IEEE Std 1003.1-2001). If any <b>-xdev</b> primary is specified, it shall apply to the entire expression even if the <b>-xdev</b> primary would not normally be evaluated.                                                                                                                                                                                                                                                                                                                                              |
| 17529 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17530 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17531 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17532 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17533 | <b>-prune</b>        | The primary shall always evaluate as true; it shall cause <i>find</i> not to descend the current pathname if it is a directory. If the <b>-depth</b> primary is specified, the <b>-prune</b> primary shall have no effect.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 17534 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17535 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17536 | <b>-perm [-]mode</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17537 |                      | The <i>mode</i> argument is used to represent file mode bits. It shall be identical in format to the <i>symbolic_mode</i> operand described in <i>chmod</i> , and shall be interpreted as follows. To start, a template shall be assumed with all file mode bits cleared. An <i>op</i> symbol of '+' shall set the appropriate mode bits in the template; '-' shall clear the appropriate bits; '=' shall set the appropriate mode bits, without regard to the contents of process' file mode creation mask. The <i>op</i> symbol of '-' cannot be the first character of <i>mode</i> ; this avoids ambiguity with the optional leading hyphen. Since the initial mode is all bits off, there are not any symbolic modes that need to use '-' as the first character. |
| 17538 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17539 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17540 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17541 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17542 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17543 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17544 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17545 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17546 |                      | If the hyphen is omitted, the primary shall evaluate as true when the file permission bits exactly match the value of the resulting template.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 17547 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17548 |                      | Otherwise, if <i>mode</i> is prefixed by a hyphen, the primary shall evaluate as true if at least all the bits in the resulting template are set in the file permission bits.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 17549 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17550 | <b>-perm [-]onum</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17551 |                      | If the hyphen is omitted, the primary shall evaluate as true when the file permission bits exactly match the value of the octal number <i>onum</i> and only the bits corresponding to the octal mask 07777 shall be compared. (See the description of the octal <i>mode</i> in <i>chmod</i> .) Otherwise, if <i>onum</i> is prefixed by a hyphen, the primary shall evaluate as true if at least all of the bits specified in <i>onum</i> that are also set in the octal mask 07777 are set.                                                                                                                                                                                                                                                                          |
| 17552 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17553 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17554 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17555 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17556 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17557 | <b>-type c</b>       | The primary shall evaluate as true if the type of the file is <i>c</i> , where <i>c</i> is 'b', 'c', 'd', 'l', 'p', 'f', or 's' for block special file, character special file, directory, symbolic link, FIFO, regular file, or socket, respectively.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 17558 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17559 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17560 | <b>-links n</b>      | The primary shall evaluate as true if the file has <i>n</i> links.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 17561 | <b>-user uname</b>   | The primary shall evaluate as true if the file belongs to the user <i>uname</i> . If <i>uname</i> is a decimal integer and the <i>getpwnam()</i> (or equivalent) function does not return a valid user name, <i>uname</i> shall be interpreted as a user ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 17562 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17563 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17564 | <b>-group gname</b>  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17565 |                      | The primary shall evaluate as true if the file belongs to the group <i>gname</i> . If <i>gname</i> is a decimal integer and the <i>getgrnam()</i> (or equivalent) function does not return a valid group name, <i>gname</i> shall be interpreted as a group ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17566 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 17567 |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

17568       **–size** *n*[*c*]     The primary shall evaluate as true if the file size in bytes, divided by 512 and  
 17569                             rounded up to the next integer, is *n*. If *n* is followed by the character '*c*', the size  
 17570                             shall be in bytes.

17571       **–atime** *n*       The primary shall evaluate as true if the file access time subtracted from the  
 17572                             initialization time, divided by 86 400 (with any remainder discarded), is *n*.

17573       **–ctime** *n*       The primary shall evaluate as true if the time of last change of file status  
 17574                             information subtracted from the initialization time, divided by 86 400 (with any  
 17575                             remainder discarded), is *n*.

17576       **–mtime** *n*       The primary shall evaluate as true if the file modification time subtracted from the  
 17577                             initialization time, divided by 86 400 (with any remainder discarded), is *n*.

17578       **–exec** *utility\_name* [*argument* ...];  
 17579       **–exec** *utility\_name* [*argument* ...] {} +  
 17580                             The end of the primary expression shall be punctuated by a semicolon or by a plus  
 17581                             sign. Only a plus sign that follows an argument containing the two characters  
 17582                             "{}" shall punctuate the end of the primary expression. Other uses of the plus  
 17583                             sign shall not be treated as special.

17584                             If the primary expression is punctuated by a semicolon, the utility *utility\_name*  
 17585                             shall be invoked once for each pathname and the primary shall evaluate as true if  
 17586                             the utility returns a zero value as exit status. A *utility\_name* or *argument* containing  
 17587                             only the two characters "{}" shall be replaced by the current pathname.

17588                             If the primary expression is punctuated by a plus sign, the primary shall always  
 17589                             evaluate as true, and the pathnames for which the primary is evaluated shall be  
 17590                             aggregated into sets. The utility *utility\_name* shall be invoked once for each set of  
 17591                             aggregated pathnames. Each invocation shall begin after the last pathname in the  
 17592                             set is aggregated, and shall be completed before the *find* utility exits and before the  
 17593                             first pathname in the next set (if any) is aggregated for this primary, but it is  
 17594                             otherwise unspecified whether the invocation occurs before, during, or after the  
 17595                             evaluations of other primaries. If any invocation returns a non-zero value as exit  
 17596                             status, the *find* utility shall return a non-zero exit status. An argument containing  
 17597                             only the two characters "{}" shall be replaced by the set of aggregated  
 17598                             pathnames, with each pathname passed as a separate argument to the invoked  
 17599                             utility in the same order that it was aggregated. The size of any set of two or more  
 17600                             pathnames shall be limited such that execution of the utility does not cause the  
 17601                             system's {ARG\_MAX} limit to be exceeded. If more than one argument containing  
 17602                             only the two characters "{}" is present, the behavior is unspecified.

17603                             If a *utility\_name* or *argument* string contains the two characters "{}", but not just  
 17604                             the two characters "{}", it is implementation-defined whether *find* replaces those  
 17605                             two characters or uses the string without change. The current directory for the  
 17606                             invocation of *utility\_name* shall be the same as the current directory when the *find*  
 17607                             utility was started. If the *utility\_name* names any of the special built-in utilities (see  
 17608                             Section 2.14 (on page 64)), the results are undefined.

17609       **–ok** *utility\_name* [*argument* ...];  
 17610                             The **–ok** primary shall be equivalent to **–exec**, except that the use of a plus sign to  
 17611                             punctuate the end of the primary expression need not be supported, and *find* shall  
 17612                             request affirmation of the invocation of *utility\_name* using the current file as an  
 17613                             argument by writing to standard error as described in the STDERR section. If the  
 17614                             response on standard input is affirmative, the utility shall be invoked. Otherwise,  
 17615                             the command shall not be invoked and the value of the **–ok** operand shall be false.

17616       **-print**       The primary shall always evaluate as true; it shall cause the current pathname to  
17617                   be written to standard output.

17618       **-newer file**   The primary shall evaluate as true if the modification time of the current file is  
17619                   more recent than the modification time of the file named by the pathname *file*.

17620       **-depth**       The primary shall always evaluate as true; it shall cause descent of the directory  
17621                   hierarchy to be done so that all entries in a directory are acted on before the  
17622                   directory itself. If a **-depth** primary is not specified, all entries in a directory shall  
17623                   be acted on after the directory itself. If any **-depth** primary is specified, it shall  
17624                   apply to the entire expression even if the **-depth** primary would not normally be  
17625                   evaluated.

17626       The primaries can be combined using the following operators (in order of decreasing  
17627       precedence):

17628       (*expression*)   True if *expression* is true.

17629       !*expression*   Negation of a primary; the unary NOT operator.

17630       *expression* [**-a**] *expression*  
17631                   Conjunction of primaries; the AND operator is implied by the juxtaposition of two  
17632                   primaries or made explicit by the optional **-a** operator. The second expression  
17633                   shall not be evaluated if the first expression is false.

17634       *expression* **-o** *expression*  
17635                   Alternation of primaries; the OR operator. The second expression shall not be  
17636                   evaluated if the first expression is true.

17637       If no *expression* is present, **-print** shall be used as the expression. Otherwise, if the given  
17638       expression does not contain any of the primaries **-exec**, **-ok**, or **-print**, the given expression shall  
17639       be effectively replaced by:

17640       (*given\_expression*) **-print**

17641       The **-user**, **-group**, and **-newer** primaries each shall evaluate their respective arguments only  
17642       once.

17643       **STDIN**  
17644       If the **-ok** primary is used, the response shall be read from the standard input. An entire line  
17645       shall be read as the response. Otherwise, the standard input shall not be used.

17646       **INPUT FILES**  
17647       None.

17648       **ENVIRONMENT VARIABLES**  
17649       The following environment variables shall affect the execution of *find*:

17650       **LANG**       Provide a default value for the internationalization variables that are unset or null.  
17651                   (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
17652                   Internationalization Variables for the precedence of internationalization variables  
17653                   used to determine the values of locale categories.)

17654       **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
17655                   internationalization variables.

17656       **LC\_COLLATE**  
17657                   Determine the locale for the behavior of ranges, equivalence classes, and multi-  
17658                   character collating elements used in the pattern matching notation for the **-n**  
17659                   option and in the extended regular expression defined for the **yesexpr** locale

17660 keyword in the *LC\_MESSAGES* category.

17661 *LC\_CTYPE* This variable determines the locale for the interpretation of sequences of bytes of  
 17662 text data as characters (for example, single-byte as opposed to multi-byte  
 17663 characters in arguments), the behavior of character classes within the pattern  
 17664 matching notation used for the **-n** option, and the behavior of character classes  
 17665 within regular expressions used in the extended regular expression defined for the  
 17666 **yesexpr** locale keyword in the *LC\_MESSAGES* category.

17667 *LC\_MESSAGES*  
 17668 Determine the locale for the processing of affirmative responses that should be  
 17669 used to affect the format and contents of diagnostic messages written to standard  
 17670 error.

17671 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

17672 *PATH* Determine the location of the *utility\_name* for the **-exec** and **-ok** primaries, as  
 17673 described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8,  
 17674 Environment Variables.

17675 **ASYNCHRONOUS EVENTS**

17676 Default.

17677 **STDOUT**

17678 The **-print** primary shall cause the current pathnames to be written to standard output. The  
 17679 format shall be:

17680 "%s\n", <path>

17681 **STDERR**

17682 The **-ok** primary shall write a prompt to standard error containing at least the *utility\_name* to be  
 17683 invoked and the current pathname. In the POSIX locale, the last non-<blank> in the prompt shall  
 17684 be ' ? '. The exact format used is unspecified.

17685 Otherwise, the standard error shall be used only for diagnostic messages.

17686 **OUTPUT FILES**

17687 None.

17688 **EXTENDED DESCRIPTION**

17689 None.

17690 **EXIT STATUS**

17691 The following exit values shall be returned:

17692 0 All *path* operands were traversed successfully.

17693 >0 An error occurred.

17694 **CONSEQUENCES OF ERRORS**

17695 Default.

17696 **APPLICATION USAGE**

17697 When used in operands, pattern matching notation, semicolons, opening parentheses, and  
 17698 closing parentheses are special to the shell and must be quoted (see Section 2.2 (on page 30)).

17699 The bit that is traditionally used for sticky (historically 01000) is specified in the **-perm** primary  
 17700 using the octal number argument form. Since this bit is not defined by this volume of  
 17701 IEEE Std 1003.1-2001, applications must not assume that it actually refers to the traditional  
 17702 sticky bit.

17703 **EXAMPLES**

17704 1. The following commands are equivalent:

```
17705 find .
17706 find . -print
```

17707 They both write out the entire directory hierarchy from the current directory.

17708 2. The following command:

```
17709 find / \(-name tmp -o -name '*.xx' \) -atime +7 -exec rm {} \;
```

17710 removes all files named **tmp** or ending in **.xx** that have not been accessed for seven or more  
 17711 24-hour periods.

17712 3. The following command:

```
17713 find . -perm -o+w,+s
```

17714 prints (**-print** is assumed) the names of all files in or below the current directory, with all  
 17715 of the file permission bits **S\_ISUID**, **S\_ISGID**, and **S\_IWOTH** set.

17716 4. The following command:

```
17717 find . -name SCCS -prune -o -print
```

17718 recursively prints pathnames of all files in the current directory and below, but skips  
 17719 directories named **SCCS** and files in them.

17720 5. The following command:

```
17721 find . -print -name SCCS -prune
```

17722 behaves as in the previous example, but prints the names of the **SCCS** directories.

17723 6. The following command is roughly equivalent to the **-nt** extension to *test*:

```
17724 if [-n "$(find file1 -prune -newer file2)"]; then
17725 printf %s\n "file1 is newer than file2"
17726 fi
```

17727 7. The descriptions of **-atime**, **-ctime**, and **-mtime** use the terminology *n* “86400 second  
 17728 periods (days)”. For example, a file accessed at 23:59 is selected by:

```
17729 find . -atime -1 -print
```

17730 at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight  
 17731 boundary between days has no effect on the 24-hour calculation.

17732 **RATIONALE**

17733 The **-a** operator was retained as an optional operator for compatibility with historical shell  
 17734 scripts, even though it is redundant with expression concatenation.

17735 The descriptions of the `'-'` modifier on the *mode* and *onum* arguments to the `-perm` primary  
 17736 agree with historical practice on BSD and System V implementations. System V and BSD  
 17737 documentation both describe it in terms of checking additional bits; in fact, it uses the same bits,  
 17738 but checks for having at least all of the matching bits set instead of having exactly the matching  
 17739 bits set.

17740 The exact format of the interactive prompts is unspecified. Only the general nature of the  
 17741 contents of prompts are specified because:

- 17742 • Implementations may desire more descriptive prompts than those used on historical  
 17743 implementations.
- 17744 • Since the historical prompt strings do not terminate with `<newline>`s, there is no portable  
 17745 way for another program to interact with the prompts of this utility via pipes.

17746 Therefore, an application using this prompting option relies on the system to provide the most  
 17747 suitable dialog directly with the user, based on the general guidelines specified.

17748 The `-name` file operand was changed to use the shell pattern matching notation so that *find* is  
 17749 consistent with other utilities using pattern matching.

17750 The `-size` operand refers to the size of a file, rather than the number of blocks it may occupy in  
 17751 the file system. The intent is that the *st\_size* field defined in the System Interfaces volume of  
 17752 IEEE Std 1003.1-2001 should be used, not the *st\_blocks* found in historical implementations. There  
 17753 are at least two reasons for this:

- 17754 1. In both System V and BSD, *find* only uses *st\_size* in size calculations for the operands  
 17755 specified by this volume of IEEE Std 1003.1-2001. (BSD uses *st\_blocks* only when processing  
 17756 the `-ls` primary.)
- 17757 2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls* utility  
 17758 for the output from the `-l` option. (In both System V and BSD, *ls* uses *st\_size* for the `-l`  
 17759 option size field and uses *st\_blocks* for the *ls -s* calculations. This volume of  
 17760 IEEE Std 1003.1-2001 does not specify *ls -s*.)

17761 The descriptions of `-atime`, `-ctime`, and `-mtime` were changed from the SVID description of *n*  
 17762 “days” to “24-hour periods”. The description is also different in terms of the exact timeframe for  
 17763 the *n* case (*versus* the `+n` or `-n`), but it matches all known historical implementations. It refers to  
 17764 one 86 400 second period in the past, not any time from the beginning of that period to the  
 17765 current time. For example, `-atime 3` is true if the file was accessed any time in the period from 72  
 17766 hours to 48 hours ago.

17767 Historical implementations do not modify `"{}"` when it appears as a substring of an `-exec` or  
 17768 `-ok utility_name` or argument string. There have been numerous user requests for this extension,  
 17769 so this volume of IEEE Std 1003.1-2001 allows the desired behavior. At least one recent  
 17770 implementation does support this feature, but encountered several problems in managing  
 17771 memory allocation and dealing with multiple occurrences of `"{}"` in a string while it was being  
 17772 developed, so it is not yet required behavior.

17773 Assuming the presence of `-print` was added to correct a historical pitfall that plagues novice  
 17774 users, it is entirely upwards-compatible from the historical System V *find* utility. In its simplest  
 17775 form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers  
 17776 agreed that adding `-print` as a default expression was the correct decision and have added the  
 17777 fast *find* functionality within a new utility called *locate*.

17778 Historically, the `-L` option was implemented using the primary `-follow`. The `-H` and `-L` options  
 17779 were added for two reasons. First, they offer a finer granularity of control and consistency with  
 17780 other programs that walk file hierarchies. Second, the `-follow` primary always evaluated to true.

As they were historically really global variables that took effect before the traversal began, some valid expressions had unexpected results. An example is the expression **-print -o -follow**. Because **-print** always evaluates to true, the standard order of evaluation implies that **-follow** would never be evaluated. This was never the case. Historical practice for the **-follow** primary, however, is not consistent. Some implementations always follow symbolic links on the command line whether **-follow** is specified or not. Others follow symbolic links on the command line only if **-follow** is specified. Both behaviors are provided by the **-H** and **-L** options, but scripts using the current **-follow** primary would be broken if the **-follow** option is specified to work either way.

Since the **-L** option resolves all symbolic links and the **-type l** primary is true for symbolic links that still exist after symbolic links have been resolved, the command:

```
find -L . -type l
```

prints a list of symbolic links reachable from the current directory that do not resolve to accessible files.

A feature of SVR4's *find* utility was the **-exec** primary's + terminator. This allowed filenames containing special characters (especially <newline>s) to be grouped together without the problems that occur if such filenames are piped to *xargs*. Other implementations have added other ways to get around this problem, notably a **-print0** primary that wrote filenames with a null byte terminator. This was considered here, but not adopted. Using a null terminator meant that any utility that was going to process *find*'s **-print0** output had to add a new option to parse the null terminators it would now be reading.

The "**-exec ... {} +**" syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210. It should be noted that this is an incompatible change to the ISO/IEC 9899:1999 standard. For example, the following command prints all files with a '-' after their name if they are regular files, and a '+' otherwise:

```
find / -type f -exec echo {} - ';' -o -exec echo {} + ';' ;
```

The change invalidates usage like this. Even though the previous standard stated that this usage would work, in practice many did not support it and the standard developers felt it better to now state that this was not allowable.

## 17810 FUTURE DIRECTIONS

17811 None.

## 17812 SEE ALSO

17813 Section 2.2 (on page 30), Section 2.13 (on page 62), Section 2.14 (on page 64), *chmod*, *pax*, *sh*, *test*,  
17814 the System Interfaces volume of IEEE Std 1003.1-2001, *getgrgid()*, *getpwuid()*, *stat()*

## 17815 CHANGE HISTORY

17816 First released in Issue 2.

## 17817 Issue 5

17818 The FUTURE DIRECTIONS section is added.

## 17819 Issue 6

17820 The following new requirements on POSIX implementations derive from alignment with the  
17821 Single UNIX Specification:

- 17822 • The **-perm [-]onum** primary is supported.

17823 The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of  
17824 symbolic links and changes to the description of the **atime**, **ctime**, and **mtime** operands.

17825

IEEE PASC Interpretation 1003.2 #210 is applied, extending the **—exec** operand.



17826 **NAME**

17827       fold — filter for folding lines

17828 **SYNOPSIS**17829       fold [-bs] [-w *width*] [*file*...]17830 **DESCRIPTION**

17831       The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a  
 17832       maximum of *width* column positions (or bytes, if the **-b** option is specified). Lines shall be  
 17833       broken by the insertion of a <newline> such that each output line (referred to later in this section  
 17834       as a *segment*) is the maximum width possible that does not exceed the specified number of  
 17835       column positions (or bytes). A line shall not be broken in the middle of a character. The behavior  
 17836       is undefined if *width* is less than the number of columns any single character in the input would  
 17837       occupy.

17838       If the <carriage-return>s, <backspace>s, or <tab>s are encountered in the input, and the **-b**  
 17839       option is not specified, they shall be treated specially:

17840       <backspace> The current count of line width shall be decremented by one, although the count  
 17841       never shall become negative. The *fold* utility shall not insert a <newline>  
 17842       immediately before or after any <backspace>.

17843       <carriage-return>

17844               The current count of line width shall be set to zero. The *fold* utility shall not insert a  
 17845       <newline> immediately before or after any <carriage-return>.

17846       <tab>       Each <tab> encountered shall advance the column position pointer to the next tab  
 17847       stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

17848 **OPTIONS**

17849       The *fold* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 17850       12.2, Utility Syntax Guidelines.

17851       The following options shall be supported:

17852       **-b**       Count *width* in bytes rather than column positions.

17853       **-s**       If a segment of a line contains a <blank> within the first *width* column positions (or  
 17854       bytes), break the line after the last such <blank> meeting the width constraints. If  
 17855       there is no <blank> meeting the requirements, the **-s** option shall have no effect for  
 17856       that output segment of the input line.

17857       **-w width**   Specify the maximum line length, in column positions (or bytes if **-b** is specified).  
 17858       The results are unspecified if *width* is not a positive decimal number. The default  
 17859       value shall be 80.

17860 **OPERANDS**

17861       The following operand shall be supported:

17862       *file*       A pathname of a text file to be folded. If no *file* operands are specified, the standard  
 17863       input shall be used.

17864 **STDIN**

17865       The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 17866       section.

17867 **INPUT FILES**

17868        If the **-b** option is specified, the input files shall be text files except that the lines are not limited  
 17869        to {LINE\_MAX} bytes in length. If the **-b** option is not specified, the input files shall be text files.

17870 **ENVIRONMENT VARIABLES**

17871        The following environment variables shall affect the execution of *fold*:

17872        **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 17873                        (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 17874                        Internationalization Variables for the precedence of internationalization variables  
 17875                        used to determine the values of locale categories.)

17876        **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 17877                        internationalization variables.

17878        **LC\_CTYPE**    Determine the locale for the interpretation of sequences of bytes of text data as  
 17879                        characters (for example, single-byte as opposed to multi-byte characters in  
 17880                        arguments and input files), and for the determination of the width in column  
 17881                        positions each character would occupy on a constant-width font output device.

17882        **LC\_MESSAGES**  
 17883                        Determine the locale that should be used to affect the format and contents of  
 17884                        diagnostic messages written to standard error.

17885 **XSI**        **NLSPATH**    Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

17886 **ASYNCHRONOUS EVENTS**

17887        Default.

17888 **STDOUT**

17889        The standard output shall be a file containing a sequence of characters whose order shall be  
 17890        preserved from the input files, possibly with inserted <newline>s.

17891 **STDERR**

17892        The standard error shall be used only for diagnostic messages.

17893 **OUTPUT FILES**

17894        None.

17895 **EXTENDED DESCRIPTION**

17896        None.

17897 **EXIT STATUS**

17898        The following exit values shall be returned:

17899        0    All input files were processed successfully.

17900        >0   An error occurred.

17901 **CONSEQUENCES OF ERRORS**

17902        Default.

**17903 APPLICATION USAGE**

17904 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths. The  
17905 *cut* utility should be used when the number of lines (or records) needs to remain constant. The  
17906 *fold* utility should be used when the contents of long lines need to be kept contiguous.

17907 The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines  
17908 wider than the printer is able to print (usually 80 or 132 column positions).

**17909 EXAMPLES**

17910 An example invocation that submits a file of possibly long lines to the printer (under the  
17911 assumption that the user knows the line width of the printer to be assigned by *lp*):

17912 `fold -w 132 bigfile | lp`

**17913 RATIONALE**

17914 Although terminal input in canonical processing mode requires the erase character (frequently  
17915 set to <backspace>) to erase the previous character (not byte or column position), terminal  
17916 output is not buffered and is extremely difficult, if not impossible, to parse correctly; the  
17917 interpretation depends entirely on the physical device that actually displays/prints/stores the  
17918 output. In all known internationalized implementations, the utilities producing output for mixed  
17919 column-width output assume that a <backspace> backs up one column position and outputs  
17920 enough <backspace>s to return to the start of the character when <backspace> is used to  
17921 provide local line motions to support underlining and emboldening operations. Since *fold*  
17922 without the **-b** option is dealing with these same constraints, <backspace> is always treated as  
17923 backing up one column position rather than backing up one character.

17924 Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column  
17925 position when written out. This is no longer always true. Since the most common usage of *fold* is  
17926 believed to be folding long lines for output to limited-length output devices, this capability was  
17927 preserved as the default case. The **-b** option was added so that applications could *fold* files with  
17928 arbitrary length lines into text files that could then be processed by the standard utilities. Note  
17929 that although the width for the **-b** option is in bytes, a line is never split in the middle of a  
17930 character. (It is unspecified what happens if a width is specified that is too small to hold a single  
17931 character found in the input followed by a <newline>.)

17932 The tab stops are hardcoded to be every eighth column to meet historical practice. No new  
17933 method of specifying other tab stops was invented.

**17934 FUTURE DIRECTIONS**

17935 None.

**17936 SEE ALSO**

17937 *cut*

**17938 CHANGE HISTORY**

17939 First released in Issue 4.

**17940 Issue 6**

17941 The normative text is reworded to avoid use of the term “must” for application requirements.

17942 **NAME**17943 fort77 — FORTRAN compiler (**FORTTRAN**)17944 **SYNOPSIS**

```
17945 FD fort77 [-c] [-g] [-L directory]... [-O optlevel] [-o outfile] [-s] [-w]
17946 operand...
```

17947

17948 **DESCRIPTION**

17949 The *fort77* utility is the interface to the FORTRAN compilation system; it shall accept the full  
 17950 FORTRAN-77 language defined by the ANSI X3.9-1978 standard. The system conceptually  
 17951 consists of a compiler and link editor. The files referenced by *operands* are compiled and linked  
 17952 to produce an executable file. It is unspecified whether the linking occurs entirely within the  
 17953 operation of *fort77*; some implementations may produce objects that are not fully resolved until  
 17954 the file is executed.

17955 If the **-c** option is present, for all pathname operands of the form *file.f*, the files:

17956 \$(basename *pathname.f*) .o

17957 shall be created or overwritten as the result of successful compilation. If the **-c** option is not  
 17958 specified, it is unspecified whether such **.o** files are created or deleted for the *file.f* operands.

17959 If there are no options that prevent link editing (such as **-c**) and all operands compile and link  
 17960 without error, the resulting executable file shall be written into the file named by the **-o** option  
 17961 (if present) or to the file **a.out**. The executable file shall be created as specified in the System  
 17962 Interfaces volume of IEEE Std 1003.1-2001, except that the file permissions shall be set to:

17963 S\_IRWXO | S\_IRWXG | S\_IRWXU

17964 and that the bits specified by the *umask* of the process shall be cleared.

17965 **OPTIONS**

17966 The *fort77* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 17967 12.2, Utility Syntax Guidelines, except that:

- 17968 • The **-l** *library* operands have the format of options, but their position within a list of  
 17969 operands affects the order in which libraries are searched.
- 17970 • The order of specifying the multiple **-L** options is significant.
- 17971 • Conforming applications shall specify each option separately; that is, grouping option letters  
 17972 (for example, **-cg**) need not be recognized by all implementations.

17973 The following options shall be supported:

17974 **-c** Suppress the link-edit phase of the compilation, and do not remove any object files  
 17975 that are produced.

17976 **-g** Produce symbolic information in the object or executable files; the nature of this  
 17977 information is unspecified, and may be modified by implementation-defined  
 17978 interactions with other options.

17979 **-s** Produce object or executable files, or both, from which symbolic and other  
 17980 information not required for proper execution using the *exec* family of functions  
 17981 defined in the System Interfaces volume of IEEE Std 1003.1-2001 has been removed  
 17982 (stripped). If both **-g** and **-s** options are present, the action taken is unspecified.

17983 **-o** *outfile* Use the pathname *outfile*, instead of the default **a.out**, for the executable file  
 17984 produced. If the **-o** option is present with **-c**, the result is unspecified.

17985        **-L** *directory*    Change the algorithm of searching for the libraries named in **-I** operands to look in  
 17986                            the directory named by the *directory* pathname before looking in the usual places.  
 17987                            Directories named in **-L** options shall be searched in the specified order. At least  
 17988                            ten instances of this option shall be supported in a single *fort77* command  
 17989                            invocation. If a directory specified by a **-L** option contains a file named **libf.a**, the  
 17990                            results are unspecified.

17991        **-O** *optlevel*    Specify the level of code optimization. If the *optlevel* option-argument is the digit  
 17992                            '0', all special code optimizations shall be disabled. If it is the digit '1', the  
 17993                            nature of the optimization is unspecified. If the **-O** option is omitted, the nature of  
 17994                            the system's default optimization is unspecified. It is unspecified whether code  
 17995                            generated in the presence of the **-O 0** option is the same as that generated when  
 17996                            **-O** is omitted. Other *optlevel* values may be supported.

17997        **-w**                Suppress warnings.

17998        Multiple instances of **-L** options can be specified.

## 17999 OPERANDS

18000        An *operand* is either in the form of a pathname or the form **-I library**. At least one operand of the  
 18001        pathname form shall be specified. The following operands shall be supported:

18002        *file.f*            The pathname of a FORTRAN source file to be compiled and optionally passed to  
 18003                            the link editor. The filename operand shall be of this form if the **-c** option is used.

18004        *file.a*            A library of object files typically produced by *ar*, and passed directly to the link  
 18005                            editor. Implementations may recognize implementation-defined suffixes other  
 18006                            than **.a** as denoting object file libraries.

18007        *file.o*            An object file produced by *fort77 -c* and passed directly to the link editor.  
 18008                            Implementations may recognize implementation-defined suffixes other than **.o** as  
 18009                            denoting object files.

18010        The processing of other files is implementation-defined.

18011        **-I library**        (The letter ell.) Search the library named:

18012                            *liblibrary.a*

18013                            A library is searched when its name is encountered, so the placement of a **-I**  
 18014                            operand is significant. Several standard libraries can be specified in this manner, as  
 18015                            described in the EXTENDED DESCRIPTION section. Implementations may  
 18016                            recognize implementation-defined suffixes other than **.a** as denoting libraries.

## 18017 STDIN

18018        Not used.

## 18019 INPUT FILES

18020        The input file shall be one of the following: a text file containing FORTRAN source code; an  
 18021        object file in the format produced by *fort77 -c*; or a library of object files, in the format produced  
 18022        by archiving zero or more object files, using *ar*. Implementations may supply additional utilities  
 18023        that produce files in these formats. Additional input files are implementation-defined.

18024        A <tab> encountered within the first six characters on a line of source code shall cause the  
 18025        compiler to interpret the following character as if it were the seventh character on the line (that  
 18026        is, in column 7).

18027 **ENVIRONMENT VARIABLES**

18028 The following environment variables shall affect the execution of *fort77*:

18029 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 18030 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 18031 Internationalization Variables for the precedence of internationalization variables  
 18032 used to determine the values of locale categories.)

18033 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 18034 internationalization variables.

18035 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 18036 characters (for example, single-byte as opposed to multi-byte characters in  
 18037 arguments and input files).

18038 **LC\_MESSAGES**  
 18039 Determine the locale that should be used to affect the format and contents of  
 18040 diagnostic messages written to standard error.

18041 **XLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

18042 **TMPDIR** Determine the pathname that should override the default directory for temporary  
 18043 files, if any.

18044 **ASYNCHRONOUS EVENTS**

18045 Default.

18046 **STDOUT**

18047 Not used.

18048 **STDERR**

18049 The standard error shall be used only for diagnostic messages. If more than one *file* operand  
 18050 ending in *.f* (or possibly other unspecified suffixes) is given, for each such file:

18051 "%s:\n", <*file*>

18052 may be written to allow identification of the diagnostic message with the appropriate input file.

18053 This utility may produce warning messages about certain conditions that do not warrant  
 18054 returning an error (non-zero) exit value.

18055 **OUTPUT FILES**

18056 Object files, listing files, and executable files shall be produced in unspecified formats.

18057 **EXTENDED DESCRIPTION**18058 **Standard Libraries**

18059 The *fort77* utility shall recognize the following **-l** operand for the standard library:

18060 **-l f** This library contains all functions referenced in the ANSI X3.9-1978 standard. This  
 18061 operand shall not be required to be present to cause a search of this library.

18062 In the absence of options that inhibit invocation of the link editor, such as **-c**, the *fort77* utility  
 18063 shall cause the equivalent of a **-l f** operand to be passed to the link editor as the last **-l** operand,  
 18064 causing it to be searched after all other object files and libraries are loaded.

18065 It is unspecified whether the library **libf.a** exists as a regular file. The implementation may  
 18066 accept as **-l** operands names of objects that do not exist as regular files.

**18067 External Symbols**

18068 The FORTRAN compiler and link editor shall support the significance of external symbols up to  
18069 a length of at least 31 bytes; case folding is permitted. The action taken upon encountering  
18070 symbols exceeding the implementation-defined maximum symbol length is unspecified.

18071 The compiler and link editor shall support a minimum of 511 external symbols per source or  
18072 object file, and a minimum of 4095 external symbols total. A diagnostic message is written to  
18073 standard output if the implementation-defined limit is exceeded; other actions are unspecified.

**18074 EXIT STATUS**

18075 The following exit values shall be returned:

18076 0 Successful compilation or link edit.

18077 >0 An error occurred.

**18078 CONSEQUENCES OF ERRORS**

18079 When *fort77* encounters a compilation error, it shall write a diagnostic to standard error and  
18080 continue to compile other source code operands. It shall return a non-zero exit status, but it is  
18081 implementation-defined whether an object module is created. If the link edit is unsuccessful, a  
18082 diagnostic message shall be written to standard error, and *fort77* shall exit with a non-zero  
18083 status.

**18084 APPLICATION USAGE**

18085 None.

**18086 EXAMPLES**

18087 The following usage example compiles **xyz.f** and creates the executable file **foo**:

18088 `fort77 -o foo xyz.f`

18089 The following example compiles **xyz.f** and creates the object file **xyz.o**:

18090 `fort77 -c xyz.f`

18091 The following example compiles **xyz.f** and creates the executable file **a.out**:

18092 `fort77 xyz.f`

18093 The following example compiles **xyz.f**, links it with **b.o**, and creates the executable **a.out**:

18094 `fort77 xyz.f b.o`

**18095 RATIONALE**

18096 The name of this utility was chosen as *fort77* to parallel the renaming of the C compiler. The  
18097 name *f77* was not chosen to avoid problems with historical implementations. The  
18098 ANSI X3.9-1978 standard was selected as a normative reference because the ISO/IEC version of  
18099 FORTRAN-77 has been superseded by the ISO/IEC 1539:1990 standard (Fortran-90).

18100 The file inclusion and symbol definition **#define** mechanisms used by the *c99* utility were not  
18101 included in this volume of IEEE Std 1003.1-2001—even though they are commonly  
18102 implemented—since there is no requirement that the FORTRAN compiler use the C  
18103 preprocessor.

18104 The **—onetrip** option was not included in this volume of IEEE Std 1003.1-2001, even though many  
18105 historical compilers support it, because it is derived from FORTRAN-66; it is an anachronism  
18106 that should not be perpetuated.

18107 Some implementations produce compilation listings. This aspect of FORTRAN has been left  
18108 unspecified because there was controversy concerning the various methods proposed for  
18109 implementing it: a **—V** option overlapped with historical vendor practice and a naming

18110 convention of creating files with **.I** suffixes collided with historical *lex* file naming practice.

18111 There is no **-I** option in this version of this volume of IEEE Std 1003.1-2001 to specify a directory  
18112 for file inclusion. An **INCLUDE** directive has been a part of the Fortran-90 discussions, but an  
18113 interface supporting that standard is not in the current scope.

18114 It is noted that many FORTRAN compilers produce an object module even when compilation  
18115 errors occur; during a subsequent compilation, the compiler may patch the object module rather  
18116 than recompiling all the code. Consequently, it is left to the implementor whether or not an  
18117 object file is created.

18118 A reference to MIL-STD-1753 was removed from an early proposal in response to a request from  
18119 the POSIX FORTRAN-binding standard developers. It was not the intention of the standard  
18120 developers to require certification of the FORTRAN compiler, and IEEE Std 1003.9-1992 does not  
18121 specify the military standard or any special preprocessing requirements. Furthermore, use of  
18122 that document would have been inappropriate for an international standard.

18123 The specification of optimization has been subject to changes through early proposals. At one  
18124 time, **-O** and **-N** were Booleans: optimize and do not optimize (with an unspecified default).  
18125 Some historical practice led this to be changed to:

18126 **-O 0** No optimization.

18127 **-O 1** Some level of optimization.

18128 **-O n** Other, unspecified levels of optimization.

18129 It is not always clear whether “good code generation” is the same thing as optimization. Simple  
18130 optimizations of local actions do not usually affect the semantics of a program. The **-O 0** option  
18131 has been included to accommodate the very particular nature of scientific calculations in a  
18132 highly optimized environment; compilers make errors. Some degree of optimization is expected,  
18133 even if it is not documented here, and the ability to shut it off completely could be important  
18134 when porting an application. An implementation may treat **-O 0** as “do less than normal” if it  
18135 wishes, but this is only meaningful if any of the operations it performs can affect the semantics  
18136 of a program. It is highly dependent on the implementation whether doing less than normal is  
18137 logical. It is not the intent of the **-O 0** option to ask for inefficient code generation, but rather to  
18138 assure that any semantically visible optimization is suppressed.

18139 The specification of standard library access is consistent with the C compiler specification.  
18140 Implementations are not required to have **/usr/lib/libf.a**, as many historical implementations do,  
18141 but if not they are required to recognize **f** as a token.

18142 External symbol size limits are in normative text; conforming applications need to know these  
18143 limits. However, the minimum maximum symbol length should be taken as a constraint on a  
18144 conforming application, not on an implementation, and consequently the action taken for a  
18145 symbol exceeding the limit is unspecified. The minimum size for the external symbol table was  
18146 added for similar reasons.

18147 The CONSEQUENCES OF ERRORS section clearly specifies the behavior of the compiler when  
18148 compilation or link-edit errors occur. The behavior of several historical implementations was  
18149 examined, and the choice was made to be silent on the status of the executable, or **a.out**, file in  
18150 the face of compiler or linker errors. If a linker writes the executable file, then links it on disk  
18151 with *lseek()*s and *write()*s, the partially linked executable file can be left on disk and its execute  
18152 bits turned off if the link edit fails. However, if the linker links the image in memory before  
18153 writing the file to disk, it need not touch the executable file (if it already exists) because the link  
18154 edit fails. Since both approaches are historical practice, a conforming application shall rely on  
18155 the exit status of *fort77*, rather than on the existence or mode of the executable file.



18156 The **-g** and **-s** options are not specified as mutually-exclusive. Historically these two options  
18157 have been mutually-exclusive, but because both are so loosely specified, it seemed appropriate  
18158 to leave their interaction unspecified.

18159 The requirement that conforming applications specify compiler options separately is to reserve  
18160 the multi-character option name space for vendor-specific compiler options, which are known to  
18161 exist in many historical implementations. Implementations are not required to recognize, for  
18162 example, **-gc** as if it were **-g -c**; nor are they forbidden from doing so. The SYNOPSIS shows all  
18163 of the options separately to highlight this requirement on applications.

18164 Echoing filenames to standard error is considered a diagnostic message because it would  
18165 otherwise be difficult to associate an error message with the erring file. They are described with  
18166 “may” to allow implementations to use other methods of identifying files and to parallel the  
18167 description in *c99*.

#### 18168 **FUTURE DIRECTIONS**

18169 A compilation system based on the ISO/IEC 1539:1990 standard (Fortran-90) may be considered  
18170 for a future version; it may have a different utility name from *fort77*.

#### 18171 **SEE ALSO**

18172 *ar*, *asa*, *c99*, *umask*, the System Interfaces volume of IEEE Std 1003.1-2001, *exec*

#### 18173 **CHANGE HISTORY**

18174 First released in Issue 4.

#### 18175 **Issue 6**

18176 This utility is marked as part of the FORTRAN Development Utilities option.

18177 The normative text is reworded to avoid use of the term “must” for application requirements.

18178 **NAME**

18179 fuser — list process IDs of all processes that have one or more files open

18180 **SYNOPSIS**

18181 xSI fuser [ -cfu ] file ...

18182

18183 **DESCRIPTION**

18184 The *fuser* utility shall write to standard output the process IDs of processes running on the local  
 18185 system that have one or more named files open. For block special devices, all processes using  
 18186 any file on that device are listed.

18187 The *fuser* utility shall write to standard error additional information about the named files  
 18188 indicating how the file is being used.

18189 Any output for processes running on remote systems that have a named file open is unspecified.

18190 A user may need appropriate privilege to invoke the *fuser* utility.

18191 **OPTIONS**

18192 The *fuser* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 18193 12.2, Utility Syntax Guidelines.

18194 The following options shall be supported:

18195 **-c** The file is treated as a mount point and the utility shall report on any files open in  
 18196 the file system.

18197 **-f** The report shall be only for the named files.

18198 **-u** The user name, in parentheses, associated with each process ID written to standard  
 18199 output shall be written to standard error.

18200 **OPERANDS**

18201 The following operand shall be supported:

18202 *file* A pathname on which the file or file system is to be reported.

18203 **STDIN**

18204 Not used.

18205 **INPUT FILES**

18206 The user database.

18207 **ENVIRONMENT VARIABLES**

18208 The following environment variables shall affect the execution of *fuser*:

18209 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 18210 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 18211 Internationalization Variables for the precedence of internationalization variables  
 18212 used to determine the values of locale categories.)

18213 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 18214 internationalization variables.

18215 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 18216 characters (for example, single-byte as opposed to multi-byte characters in  
 18217 arguments).

18218 **LC\_MESSAGES**

18219 Determine the locale that should be used to affect the format and contents of  
 18220 diagnostic messages written to standard error.

- 18221 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 18222 **ASYNCHRONOUS EVENTS**
- 18223 Default.
- 18224 **STDOUT**
- 18225 The *fuser* utility shall write the process ID for each process using each file given as an operand to
- 18226 standard output in the following format:
- 18227 "%d", <*process\_id*>
- 18228 **STDERR**
- 18229 The *fuser* utility shall write diagnostic messages to standard error.
- 18230 The *fuser* utility also shall write the following to standard error:
- 18231 • The pathname of each named file is written followed immediately by a colon.
- 18232 • For each process ID written to standard output, the character 'c' shall be written to
- 18233 standard error if the process is using the file as its current directory and the character 'r'
- 18234 shall be written to standard error if the process is using the file as its root directory.
- 18235 Implementations may write other alphabetic characters to indicate other uses of files.
- 18236 • When the *-u* option is specified, characters indicating the use of the file shall be followed
- 18237 immediately by the user name, in parentheses, corresponding to the process' real user ID. If
- 18238 the user name cannot be resolved from the process' real user ID, the process' real user ID
- 18239 shall be written instead of the user name.
- 18240 When standard output and standard error are directed to the same file, the output shall be
- 18241 interleaved so that the filename appears at the start of each line, followed by the process ID and
- 18242 characters indicating the use of the file. Then, if the *-u* option is specified, the user name or user
- 18243 ID for each process using that file shall be written.
- 18244 A <newline> shall be written to standard error after the last output described above for each *file*
- 18245 operand.
- 18246 **OUTPUT FILES**
- 18247 None.
- 18248 **EXTENDED DESCRIPTION**
- 18249 None.
- 18250 **EXIT STATUS**
- 18251 The following exit values shall be returned:
- 18252 0 Successful completion.
- 18253 >0 An error occurred.
- 18254 **CONSEQUENCES OF ERRORS**
- 18255 Default.

18256 **APPLICATION USAGE**

18257           None.

18258 **EXAMPLES**

18259           The command:

18260           `fuser -fu .`

18261           writes to standard output the process IDs of processes that are using the current directory and

18262           writes to standard error an indication of how those processes are using the directory and the

18263           user names associated with the processes that are using the current directory.

18264 **RATIONALE**18265           The definition of the *fuser* utility follows existing practice.18266 **FUTURE DIRECTIONS**

18267           None.

18268 **SEE ALSO**

18269           None.

18270 **CHANGE HISTORY**

18271           First released in Issue 5.

18272 **NAME**

18273 gencat — generate a formatted message catalog

18274 **SYNOPSIS**18275 xSI gencat *catfile* *msgfile*...

18276

18277 **DESCRIPTION**

18278 The *gencat* utility shall merge the message text source file *msgfile* into a formatted message  
 18279 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its  
 18280 messages shall be included in the new *catfile*. If set and message numbers collide, the new  
 18281 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

18282 **OPTIONS**

18283 None.

18284 **OPERANDS**

18285 The following operands shall be supported:

18286 *catfile* A pathname of the formatted message catalog. If '-' is specified, standard output  
 18287 shall be used. The format of the message catalog produced is unspecified.

18288 *msgfile* A pathname of a message text source file. If '-' is specified for an instance of  
 18289 *msgfile*, standard input shall be used. The format of message text source files is  
 18290 defined in the EXTENDED DESCRIPTION section.

18291 **STDIN**18292 The standard input shall not be used unless a *msgfile* operand is specified as '-'.18293 **INPUT FILES**

18294 The input files shall be text files.

18295 **ENVIRONMENT VARIABLES**18296 The following environment variables shall affect the execution of *gencat*:

18297 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 18298 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 18299 Internationalization Variables for the precedence of internationalization variables  
 18300 used to determine the values of locale categories.)

18301 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 18302 internationalization variables.

18303 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 18304 characters (for example, single-byte as opposed to multi-byte characters in  
 18305 arguments and input files).

18306 *LC\_MESSAGES*

18307 Determine the locale that should be used to affect the format and contents of  
 18308 diagnostic messages written to standard error.

18309 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

18310 **ASYNCHRONOUS EVENTS**

18311 Default.

18312 **STDOUT**18313 The standard output shall not be used unless the *catfile* operand is specified as '-'.

18314 **STDERR**

18315 The standard error shall be used only for diagnostic messages.

18316 **OUTPUT FILES**

18317 None.

18318 **EXTENDED DESCRIPTION**

18319 The content of a message text file shall be in the format defined as follows. Note that the fields of  
 18320 a message text source line are separated by a single <blank>. Any other <blank>s are considered  
 18321 to be part of the subsequent field.

18322 **\$set** *n comment*

18323 This line specifies the set identifier of the following messages until the next **\$set** or  
 18324 end-of-file appears. The *n* denotes the set identifier, which is defined as a number  
 18325 in the range [1, {NL\_SETMAX}] (see the <limits.h> header defined in the Base  
 18326 Definitions volume of IEEE Std 1003.1-2001). The application shall ensure that set  
 18327 identifiers are presented in ascending order within a single source file, but need  
 18328 not be contiguous. Any string following the set identifier shall be treated as a  
 18329 comment. If no **\$set** directive is specified in a message text source file, all messages  
 18330 shall be located in an implementation-defined default message set NL\_SETD (see  
 18331 the <nl\_types.h> header defined in the Base Definitions volume of  
 18332 IEEE Std 1003.1-2001).

18333 **\$delset** *n comment*

18334 This line deletes message set *n* from an existing message catalog. The *n* denotes the  
 18335 set number [1, {NL\_SETMAX}]. Any string following the set number shall be  
 18336 treated as a comment.

18337 **\$comment** A line beginning with ' \$ ' followed by a <blank> shall be treated as a comment.

18338 *m message-text*

18339 The *m* denotes the message identifier, which is defined as a number in the range [1,  
 18340 {NL\_MSGMAX}] (see the <limits.h> header). The *message-text* shall be stored in the  
 18341 message catalog with the set identifier specified by the last **\$set** directive, and with  
 18342 message identifier *m*. If the *message-text* is empty, and a <blank> field separator is  
 18343 present, an empty string shall be stored in the message catalog. If a message source  
 18344 line has a message number, but neither a field separator nor *message-text*, the  
 18345 existing message with that number (if any) shall be deleted from the catalog. The  
 18346 application shall ensure that message identifiers are in ascending order within a  
 18347 single set, but need not be contiguous. The application shall ensure that the length  
 18348 of *message-text* is in the range [0, {NL\_TEXTMAX}] (see the <limits.h> header).

18349 **\$quote** *n* This line specifies an optional quote character *c*, which can be used to surround  
 18350 *message-text* so that trailing spaces or null (empty) messages are visible in a  
 18351 message source line. By default, or if an empty **\$quote** directive is supplied, no  
 18352 quoting of *message-text* shall be recognized.

18353 Empty lines in a message text source file shall be ignored. The effects of lines starting with any  
 18354 character other than those defined above are implementation-defined.

18355 Text strings can contain the special characters and escape sequences defined in the following  
 18356 table:

18357  
18358  
18359  
18360  
18361  
18362  
18363  
18364  
18365  
18366

| Description       | Symbol | Sequence |
|-------------------|--------|----------|
| <newline>         | NL(LF) | \n       |
| Horizontal-tab    | HT     | \t       |
| <vertical-tab>    | VT     | \v       |
| <backspace>       | BS     | \b       |
| <carriage-return> | CR     | \r       |
| <form-feed>       | FF     | \f       |
| Backslash         | \      | \\       |
| Bit pattern       | ddd    | \ddd     |

18367 The escape sequence "\ddd" consists of backslash followed by one, two, or three octal digits,  
18368 which shall be taken to specify the value of the desired character. If the character following a  
18369 backslash is not one of those specified, the backslash shall be ignored.

18370 Backslash ('\'') followed by a <newline> is also used to continue a string on the following line.  
18371 Thus, the following two lines describe a single message string:

18372 1 This line continues \  
18373 to the next line

18374 which shall be equivalent to:

18375 1 This line continues to the next line

#### 18376 EXIT STATUS

18377 The following exit values shall be returned:

18378 0 Successful completion.

18379 >0 An error occurred.

#### 18380 CONSEQUENCES OF ERRORS

18381 Default.

#### 18382 APPLICATION USAGE

18383 Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot  
18384 be guaranteed between different types of machine. Thus, just as C programs need to be  
18385 recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

#### 18386 EXAMPLES

18387 None.

#### 18388 RATIONALE

18389 None.

#### 18390 FUTURE DIRECTIONS

18391 None.

#### 18392 SEE ALSO

18393 *iconv*, the Base Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <nl\_types.h>

#### 18394 CHANGE HISTORY

18395 First released in Issue 3.

#### 18396 Issue 6

18397 The normative text is reworded to avoid use of the term “must” for application requirements.

18398 **NAME**18399       get — get a version of an SCCS file (**DEVELOPMENT**)18400 **SYNOPSIS**18401 xSI       get [-begkmnlPst] [-c *cutoff*] [-i *list*] [-r *SID*] [-x *list*] *file...*

18402

18403 **DESCRIPTION**18404       The *get* utility shall generate a text file from each named SCCS *file* according to the specifications  
18405       given by its options.18406       The generated text shall normally be written into a file called the **g-file** whose name is derived  
18407       from the SCCS filename by simply removing the leading "s.". 18408 **OPTIONS**18409       The *get* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
18410       Utility Syntax Guidelines.

18411       The following options shall be supported:

18412       **-r *SID***       Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file  
18413       to be retrieved. The table shows, for the most useful cases, what version of an  
18414       SCCS file is retrieved (as well as the SID of the version to be eventually created by  
18415       *delta* if the **-e** option is also used), as a function of the SID specified.18416       **-c *cutoff***       Indicate the *cutoff* date-time, in the form:

18417               YY[MM[DD[HH[MM[SS]]]]]

18418       For the YY component, values in the range [69,99] shall refer to years 1969 to 1999  
18419       inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.18420       **Note:**       It is expected that in a future version of IEEE Std 1003.1-2001 the default  
18421       century inferred from a 2-digit year will change. (This would apply to all  
18422       commands accepting a 2-digit year as input.)18423       No changes (deltas) to the SCCS file that were created after the specified *cutoff*  
18424       date-time shall be included in the generated text file. Units omitted from the date-  
18425       time default to their maximum possible values; for example, **-c** 7502 is equivalent  
18426       to **-c** 750228235959.18427       Any number of non-numeric characters may separate the various 2-digit pieces of  
18428       the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:  
18429       **-c** "77/2/2 9:22:25".18430       **-e**       Indicate that the *get* is for the purpose of editing or making a change (delta) to the  
18431       SCCS file via a subsequent use of *delta*. The **-e** option used in a *get* for a particular  
18432       version (SID) of the SCCS file shall prevent further *get* commands from editing on  
18433       the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.  
18434       Concurrent use of *get* **-e** for different SIDs is always allowed.18435       If the **g-file** generated by *get* with a **-e** option is accidentally ruined in the process  
18436       of editing, it may be regenerated by re-executing the *get* command with the **-k**  
18437       option in place of the **-e** option.18438       SCCS file protection specified via the ceiling, floor, and authorized user list stored  
18439       in the SCCS file shall be enforced when the **-e** option is used.18440       **-b**       Use with the **-e** option to indicate that the new delta should have an SID in a new  
18441       branch as shown in the table below. This option shall be ignored if the **b** flag is not  
18442       present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that



18443 has no successors on the SCCS file tree.)

18444 **Note:** A branch delta may always be created from a non-leaf delta.

18445 **-i list** Indicate a *list* of deltas to be included (forced to be applied) in the creation of the  
 18446 generated file. The *list* has the following syntax:

18447 `<list> ::= <range> | <list> , <range>`  
 18448 `<range> ::= SID | SID - SID`

18449 SID, the SCCS Identification of a delta, may be in any form shown in the “SID  
 18450 Specified” column of the table in the EXTENDED DESCRIPTION section, except  
 18451 that the result of supplying a partial SID is unspecified. A diagnostic message shall  
 18452 be written if the first SID in the range is not an ancestor of the second SID in the  
 18453 range.

18454 **-x list** Indicate a *list* of deltas to be excluded (forced not to be applied) in the creation of  
 18455 the generated file. See the **-i** option for the *list* format.

18456 **-k** Suppress replacement of identification keywords (see below) in the retrieved text  
 18457 by their value. The **-k** option shall be implied by the **-e** option.

18458 **-l** Write a delta summary into an **l-file**.

18459 **-L** Write a delta summary to standard output. All informative output that normally is  
 18460 written to standard output shall be written to standard error instead, unless the **-s**  
 18461 option is used, in which case it shall be suppressed.

18462 **-p** Write the text retrieved from the SCCS file to the standard output. No **g-file** shall  
 18463 be created. All informative output that normally goes to the standard output shall  
 18464 go to standard error instead, unless the **-s** option is used, in which case it shall  
 18465 disappear.

18466 **-s** Suppress all informative output normally written to standard output. However,  
 18467 fatal error messages (which shall always be written to the standard error) shall  
 18468 remain unaffected.

18469 **-m** Precede each text line retrieved from the SCCS file by the SID of the delta that  
 18470 inserted the text line in the SCCS file. The format shall be:

18471 `"%s\t%s", <SID>, <text line>`

18472 **-n** Precede each generated text line with the **%M%** identification keyword value (see  
 18473 below). The format shall be:

18474 `"%s\t%s", <%M% value>, <text line>`

18475 When both the **-m** and **-n** options are used, the `<text line>` shall be replaced by the  
 18476 **-m** option-generated format.

18477 **-g** Suppress the actual retrieval of text from the SCCS file. It is primarily used to  
 18478 generate an **l-file**, or to verify the existence of a particular SID.

18479 **-t** Use to access the most recently created (top) delta in a given release (for example,  
 18480 **-r 1**), or release and level (for example, **-r 1.2**).

## 18481 OPERANDS

18482 The following operands shall be supported:

18483 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get*  
 18484 utility shall behave as though each file in the directory were specified as a named  
 18485 file, except that non-SCCS files (last component of the pathname does not begin

18486 with **s**.) and unreadable files shall be silently ignored.

18487 If exactly one *file* operand appears, and it is **'-'**, the standard input shall be read;  
 18488 each line of the standard input is taken to be the name of an SCCS file to be  
 18489 processed. Non-SCCS files and unreadable files shall be silently ignored.

18490 **STDIN**

18491 The standard input shall be a text file used only if the *file* operand is specified as **'-'**. Each line  
 18492 of the text file shall be interpreted as an SCCS pathname.

18493 **INPUT FILES**

18494 The SCCS files shall be files of an unspecified format.

18495 **ENVIRONMENT VARIABLES**

18496 The following environment variables shall affect the execution of *get*:

18497 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 18498 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 18499 Internationalization Variables for the precedence of internationalization variables  
 18500 used to determine the values of locale categories.)

18501 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 18502 internationalization variables.

18503 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 18504 characters (for example, single-byte as opposed to multi-byte characters in  
 18505 arguments and input files).

18506 **LC\_MESSAGES**

18507 Determine the locale that should be used to affect the format and contents of  
 18508 diagnostic messages written to standard error, and informative messages written  
 18509 to standard output (or standard error, if the **-p** option is used).

18510 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

18511 **TZ** Determine the timezone in which the times and dates written in the SCCS file are  
 18512 evaluated. If the *TZ* variable is unset or NULL, an unspecified system default  
 18513 timezone is used.

18514 **ASYNCHRONOUS EVENTS**

18515 Default.

18516 **STDOUT**

18517 For each file processed, *get* shall write to standard output the SID being accessed and the number  
 18518 of lines retrieved from the SCCS file, in the following format:

18519 **"%s\n%d lines\n", <SID>, <number of lines>**

18520 If the **-e** option is used, the SID of the delta to be made shall appear after the SID accessed and  
 18521 before the number of lines generated, in the POSIX locale:

18522 **"%s\nnew delta %s\n%d lines\n", <SID accessed>,  
 18523 <SID to be made>, <number of lines>**

18524 If there is more than one named file or if a directory or standard input is named, each pathname  
 18525 shall be written before each of the lines shown in one of the preceding formats:

18526 **"\n%s:\n", <pathname>**

18527 If the **-L** option is used, a delta summary shall be written following the format specified below  
 18528 for **l-files**.

18529 If the **-i** option is used, included deltas shall be listed following the notation, in the POSIX locale:

18530 "Included:\n"

18531 If the **-x** option is used, excluded deltas shall be listed following the notation, in the POSIX

18532 locale:

18533 "Excluded:\n"

18534 If the **-p** or **-L** options are specified, the standard output shall consist of the text retrieved from

18535 the SCCS file.

18536 **STDERR**

18537 The standard error shall be used only for diagnostic messages, except if the **-p** or **-L** options are

18538 specified, it shall include all informative messages normally sent to standard output.

18539 **OUTPUT FILES**

18540 Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-**

18541 **file**, **p-file**, and **z-file**. The letter before the hyphen is called the *tag*. An auxiliary filename shall

18542 be formed from the SCCS filename: the application shall ensure that the last component of all

18543 SCCS filenames is of the form **s.module-name**; the auxiliary files shall be named by replacing the

18544 leading **s** with the tag. The **g-file** shall be an exception to this scheme: the **g-file** is named by

18545 removing the **s.** prefix. For example, for **s.xyz.c**, the auxiliary filenames would be **xyz.c**, **l.xyz.c**,

18546 **p.xyz.c**, and **z.xyz.c**, respectively.

18547 The **g-file**, which contains the generated text, shall be created in the current directory (unless the

18548 **-p** option is used). A **g-file** shall be created in all cases, whether or not any lines of text were

18549 generated by the *get*. It shall be owned by the real user. If the **-k** option is used or implied, the

18550 **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be

18551 read-only. Only the real user need have write permission in the current directory.

18552 The **l-file** shall contain a table showing which deltas were applied in generating the retrieved

18553 text. The **l-file** shall be created in the current directory if the **-l** option is used; it shall be read-

18554 only and it is owned by the real user. Only the real user need have write permission in the

18555 current directory.

18556 Lines in the **l-file** shall have the following format:

18557 "%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>,

18558 <SID>, <date-time>, <login>

18559 where the entries are:

18560 <code1> A <space> if the delta was applied; ' \* ' otherwise.

18561 <code2> A <space> if the delta was applied or was not applied and ignored; ' \* ' if the delta

18562 was not applied and was not ignored.

18563 <code3> A character indicating a special reason why the delta was or was not applied:

18564 **I** Included.

18565 **X** Excluded.

18566 **C** Cut off (by a **-c** option).

18567 <date-time> Date and time (using the format of the *date* utility's %Y/%m/%d %T conversion

18568 specification format) of creation.

18569 <login> Login name of person who created *delta*.

18570 The comments and MR data shall follow on subsequent lines, indented one <tab>. A blank line  
 18571 shall terminate each entry.

18572 The **p-file** shall be used to pass information resulting from a *get* with a **-e** option along to *delta*.  
 18573 Its contents shall also be used to prevent a subsequent execution of *get* with a **-e** option for the  
 18574 same SID until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be  
 18575 created in the directory containing the SCCS file and the application shall ensure that the  
 18576 effective user has write permission in that directory. It shall be writable by owner only, and  
 18577 owned by the effective user. Each line in the **p-file** shall have the following format:

18578 "%sΔ%sΔ%sΔ%s%s\n", <g-file SID>,  
 18579 <SID of new delta>, <login-name of real user>,  
 18580 <date-time>, <i-value>, <x-value>

18581 where <i-value> uses the format " " if no **-i** option was specified, and shall use the format:

18582 "Δ-i%s", <-i option option-argument>

18583 if a **-i** option was specified and <x-value> uses the format " " if no **-x** option was specified, and  
 18584 shall use the format:

18585 "Δ-x%s", <-x option option-argument>

18586 if a **-x** option was specified. There can be an arbitrary number of lines in the **p-file** at any time;  
 18587 no two lines shall have the same new delta SID.

18588 The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall  
 18589 be the binary process ID of the command (that is, *get*) that created it. The **z-file** shall be created  
 18590 in the directory containing the SCCS file for the duration of *get*. The same protection restrictions  
 18591 as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only.

## 18592 EXTENDED DESCRIPTION

18593

18594

18595

18596

18597

18598

18599

18600

18601

18602

18603

18604

18605

18606

18607

18608

18609

18610

18611

18612

18613

18614

| Determination of SCCS Identification String |                    |                                             |               |                            |
|---------------------------------------------|--------------------|---------------------------------------------|---------------|----------------------------|
| SID* Specified                              | –b Keyletter Used† | Other Conditions                            | SID Retrieved | SID of Delta to be Created |
| none‡                                       | no                 | R defaults to mR                            | mR.mL         | mR.(mL+1)                  |
| none‡                                       | yes                | R defaults to mR                            | mR.mL         | mR.mL.(mB+1).1             |
| R                                           | no                 | R > mR                                      | mR.mL         | R.1***                     |
| R                                           | no                 | R = mR                                      | mR.mL         | mR.(mL+1)                  |
| R                                           | yes                | R > mR                                      | mR.mL         | mR.mL.(mB+1).1             |
| R                                           | yes                | R = mR                                      | mR.mL         | mR.mL.(mB+1).1             |
| R                                           | –                  | R < mR and R does not exist                 | hR.mL**       | hR.mL.(mB+1).1             |
| R                                           | –                  | Trunk successor in release > R and R exists | R.mL          | R.mL.(mB+1).1              |
| R.L                                         | no                 | No trunk successor                          | R.L           | R.(L+1)                    |
| R.L                                         | yes                | No trunk successor                          | R.L           | R.L.(mB+1).1               |
| R.L                                         | –                  | Trunk successor in release ≥ R              | R.L           | R.L.(mB+1).1               |
| R.L.B                                       | no                 | No branch successor                         | R.L.B.mS      | R.L.B.(mS+1)               |
| R.L.B                                       | yes                | No branch successor                         | R.L.B.mS      | R.L.(mB+1).1               |
| R.L.B.S                                     | no                 | No branch successor                         | R.L.B.S       | R.L.B.(S+1)                |
| R.L.B.S                                     | yes                | No branch successor                         | R.L.B.S       | R.L.(mB+1).1               |
| R.L.B.S                                     | –                  | Branch successor                            | R.L.B.S       | R.L.(mB+1).1               |

18615

18616

18617

18618

18619

18620

\* R, L, B, and S are the release, level, branch, and sequence components of the SID, respectively; m means maximum. Thus, for example, R.mL means “the maximum level number within release R”; R.L.(mB+1).1 means “the first sequence number on the new branch (that is, maximum branch number plus one) of level L within release R”. Note that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified components shall exist.

18621

\*\* hR is the highest existing release that is lower than the specified, nonexistent, release R.

18622

\*\*\* This is used to force creation of the first delta in a new release.

18623

18624

† The –b option is effective only if the b flag is present in the file. An entry of ‘–’ means “irrelevant”.

18625

18626

18627

‡ This case applies if the d (default SID) flag is not present in the file. If the d flag is present in the file, then the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

18628 **System Date and Time**

18629 When a **g-file** is generated, the creation time of deltas in the SCCS file may be taken into  
 18630 account. If any of these times are apparently in the future, the behavior is unspecified.

18631 **Identification Keywords**

18632 Identifying information shall be inserted into the text retrieved from the SCCS file by replacing  
 18633 identification keywords with their value wherever they occur. The following keywords may be  
 18634 used in the text stored in an SCCS file:

|       |            |                                                                                               |
|-------|------------|-----------------------------------------------------------------------------------------------|
| 18635 | <b>%M%</b> | Module name: either the value of the <b>m</b> flag in the file, or if absent, the name of the |
| 18636 |            | SCCS file with the leading <b>s.</b> removed.                                                 |
| 18637 | <b>%I%</b> | SCCS identification (SID) ( <b>%R%.%L%</b> or <b>%R%.%L%.%B%.%S%</b> ) of the retrieved       |
| 18638 |            | text.                                                                                         |
| 18639 | <b>%R%</b> | Release.                                                                                      |
| 18640 | <b>%L%</b> | Level.                                                                                        |
| 18641 | <b>%B%</b> | Branch.                                                                                       |
| 18642 | <b>%S%</b> | Sequence.                                                                                     |
| 18643 | <b>%D%</b> | Current date ( <i>YY/MM/DD</i> ).                                                             |
| 18644 | <b>%H%</b> | Current date ( <i>MM/DD/YY</i> ).                                                             |
| 18645 | <b>%T%</b> | Current time ( <i>HH:MM:SS</i> ).                                                             |
| 18646 | <b>%E%</b> | Date newest applied delta was created ( <i>YY/MM/DD</i> ).                                    |
| 18647 | <b>%G%</b> | Date newest applied delta was created ( <i>MM/DD/YY</i> ).                                    |
| 18648 | <b>%U%</b> | Time newest applied delta was created ( <i>HH:MM:SS</i> ).                                    |
| 18649 | <b>%Y%</b> | Module type: value of the <b>t</b> flag in the SCCS file.                                     |
| 18650 | <b>%F%</b> | SCCS filename.                                                                                |
| 18651 | <b>%P%</b> | SCCS absolute pathname.                                                                       |
| 18652 | <b>%Q%</b> | The value of the <b>q</b> flag in the file.                                                   |
| 18653 | <b>%C%</b> | Current line number. This keyword is intended for identifying messages output by              |
| 18654 |            | the program, such as “this should not have happened” type errors. It is not                   |
| 18655 |            | intended to be used on every line to provide sequence numbers.                                |
| 18656 | <b>%Z%</b> | The four-character string "@(#)" recognizable by <i>what</i> .                                |
| 18657 | <b>%W%</b> | A shorthand notation for constructing <i>what</i> strings:                                    |
| 18658 |            | <code>%W%=%Z%%M%&lt;tab&gt;%I%</code>                                                         |
| 18659 | <b>%A%</b> | Another shorthand notation for constructing <i>what</i> strings:                              |
| 18660 |            | <code>%A%=%Z%%Y%%M%%I%%Z%</code>                                                              |

18661 **EXIT STATUS**

18662 The following exit values shall be returned:

18663     0   Successful completion.

18664     >0   An error occurred.

18665 **CONSEQUENCES OF ERRORS**

18666 Default.

18667 **APPLICATION USAGE**

18668 Problems can arise if the system date and time have been modified (for example, put forward  
 18669 and then back again, or unsynchronized clocks across a network) and can also arise when  
 18670 different values of the *TZ* environment variable are used.

18671 Problems of a similar nature can also arise for the operation of the *delta* utility, which compares  
 18672 the previous file body against the working file as part of its normal operation.

18673 **EXAMPLES**

18674 None.

18675 **RATIONALE**

18676 None.

18677 **FUTURE DIRECTIONS**18678 The **-lp** option may be withdrawn in a future version.18679 **SEE ALSO**18680 *admin, delta, prs, what*18681 **CHANGE HISTORY**

18682 First released in Issue 2.

18683 **Issue 5**

18684 A correction is made to the first format string in STDOUT.

18685 The interpretation of the *YY* component of the **-c cutoff** argument is noted.18686 **Issue 6**18687 The obsolescent SYNOPSIS is removed, removing the **-lp** option.

18688 The normative text is reworded to avoid use of the term “must” for application requirements.

18689 The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

18690 The Open Group Corrigendum U048/1 is applied.

18691 The Open Group Interpretation PIN4C.00014 is applied.

18692 The Open Group Base Resolution bwg2001-007 is applied as follows:

18693 • The EXTENDED DESCRIPTION section is updated to make partial SID handling  
 18694 unspecified, reflecting common usage, and to clarify SID ranges.

18695 • New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections  
 18696 regarding how the system date and time may be taken into account.

18697 • The *TZ* environment variable is added to the ENVIRONMENT VARIABLES section.

18698 **NAME**

18699       getconf — get configuration values

18700 **SYNOPSIS**18701       getconf [ -v specification ] *system\_var*18702       getconf [ -v specification ] *path\_var* *pathname*18703 **DESCRIPTION**18704       In the first synopsis form, the *getconf* utility shall write to the standard output the value of the  
18705       variable specified by the *system\_var* operand.18706       In the second synopsis form, the *getconf* utility shall write to the standard output the value of the  
18707       variable specified by the *path\_var* operand for the path specified by the *pathname* operand.18708       The value of each configuration variable shall be determined as if it were obtained by calling the  
18709       function from which it is defined to be available by this volume of IEEE Std 1003.1-2001 or by the  
18710       System Interfaces volume of IEEE Std 1003.1-2001 (see the OPERANDS section). The value shall  
18711       reflect conditions in the current operating environment.18712 **OPTIONS**18713       The *getconf* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
18714       12.2, Utility Syntax Guidelines.

18715       The following option shall be supported:

18716       -v *specification*18717               Indicate a specific specification and version for which configuration variables shall  
18718               be determined. If this option is not specified, the values returned correspond to an  
18719               implementation default conforming compilation environment.

18720       If the command:

18721       getconf \_POSIX\_V6\_ILP32\_OFF32

18722       does not write "-1\n" or "undefined\n" to standard output, then commands of  
18723       the form:

18724       getconf -v POSIX\_V6\_ILP32\_OFF32 ...

18725       determine values for configuration variables corresponding to the  
18726       POSIX\_V6\_ILP32\_OFF32 compilation environment specified in c99, the  
18727       EXTENDED DESCRIPTION.

18728       If the command:

18729       getconf \_POSIX\_V6\_ILP32\_OFFBIG

18730       does not write "-1\n" or "undefined\n" to standard output, then commands of  
18731       the form:

18732       getconf -v POSIX\_V6\_ILP32\_OFFBIG ...

18733       determine values for configuration variables corresponding to the  
18734       POSIX\_V6\_ILP32\_OFFBIG compilation environment specified in c99, the  
18735       EXTENDED DESCRIPTION.

18736       If the command:

18737       getconf \_POSIX\_V6\_LP64\_OFF64

18738       does not write "-1\n" or "undefined\n" to standard output, then commands of  
18739       the form:



18740 `getconf -v POSIX_V6_LP64_OFF64 ...`

18741 determine values for configuration variables corresponding to the

18742 `POSIX_V6_LP64_OFF64` compilation environment specified in `c99`, the

18743 EXTENDED DESCRIPTION.

18744 If the command:

18745 `getconf _POSIX_V6_LPBIG_OFFBIG`

18746 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of

18747 the form:

18748 `getconf -v POSIX_V6_LPBIG_OFFBIG ...`

18749 determine values for configuration variables corresponding to the

18750 `POSIX_V6_LPBIG_OFFBIG` compilation environment specified in `c99`, the

18751 EXTENDED DESCRIPTION.

## 18752 OPERANDS

18753 The following operands shall be supported:

18754 *path\_var* A name of a configuration variable. All of the variables in the Variable column of  
 18755 the table in the DESCRIPTION of the *fpathconf()* function defined in the System  
 18756 Interfaces volume of IEEE Std 1003.1-2001, without the enclosing braces, shall be  
 18757 supported. The implementation may add other local variables.

18758 *pathname* A pathname for which the variable specified by *path\_var* is to be determined.

18759 *system\_var* A name of a configuration variable. All of the following variables shall be  
 18760 supported:

- The names in the Variable column of the table in the DESCRIPTION of the  
 18761 *sysconf()* function in the System Interfaces volume of IEEE Std 1003.1-2001,  
 18762 except for the entries corresponding to `_SC_CLK_TCK`,  
 18763 `_SC_GETGR_R_SIZE_MAX`, and `_SC_GETPW_R_SIZE_MAX`, without the  
 18764 enclosing braces.  
 18765

18766 For compatibility with earlier versions, the following variable names shall also  
 18767 be supported:

18768 `POSIX2_C_BIND`  
 18769 `POSIX2_C_DEV`  
 18770 `POSIX2_CHAR_TERM`  
 18771 `POSIX2_FORT_DEV`  
 18772 `POSIX2_FORT_RUN`  
 18773 `POSIX2_LOCALEDEF`  
 18774 `POSIX2_SW_DEV`  
 18775 `POSIX2_UPE`  
 18776 `POSIX2_VERSION`

18777 and shall be equivalent to the same name prefixed with an underscore. This  
 18778 requirement may be removed in a future version.

- The names of the symbolic constants used as the *name* argument of the *confstr()*  
 18779 function in the System Interfaces volume of IEEE Std 1003.1-2001, without the  
 18780 `_CS_` prefix.  
 18781

- The names of the symbolic constants listed under the headings “Maximum  
 18782 Values” and “Minimum Values” in the description of the **<limits.h>** header in  
 18783

|       |                              |                                                                                                          |  |
|-------|------------------------------|----------------------------------------------------------------------------------------------------------|--|
| 18784 |                              | the Base Definitions volume of IEEE Std 1003.1-2001, without the enclosing                               |  |
| 18785 |                              | braces.                                                                                                  |  |
| 18786 |                              | For compatibility with earlier versions, the following variable names shall also                         |  |
| 18787 |                              | be supported:                                                                                            |  |
| 18788 |                              | POSIX2_BC_BASE_MAX                                                                                       |  |
| 18789 |                              | POSIX2_BC_DIM_MAX                                                                                        |  |
| 18790 |                              | POSIX2_BC_SCALE_MAX                                                                                      |  |
| 18791 |                              | POSIX2_BC_STRING_MAX                                                                                     |  |
| 18792 |                              | POSIX2_COLL_WEIGHTS_MAX                                                                                  |  |
| 18793 |                              | POSIX2_EXPR_NEST_MAX                                                                                     |  |
| 18794 |                              | POSIX2_LINE_MAX                                                                                          |  |
| 18795 |                              | POSIX2_RE_DUP_MAX                                                                                        |  |
| 18796 |                              | and shall be equivalent to the same name prefixed with an underscore. This                               |  |
| 18797 |                              | requirement may be removed in a future version.                                                          |  |
| 18798 |                              | The implementation may add other local values.                                                           |  |
| 18799 | <b>STDIN</b>                 |                                                                                                          |  |
| 18800 |                              | Not used.                                                                                                |  |
| 18801 | <b>INPUT FILES</b>           |                                                                                                          |  |
| 18802 |                              | None.                                                                                                    |  |
| 18803 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                          |  |
| 18804 |                              | The following environment variables shall affect the execution of <i>getconf</i> :                       |  |
| 18805 | <i>LANG</i>                  | Provide a default value for the internationalization variables that are unset or null.                   |  |
| 18806 |                              | (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,                                   |  |
| 18807 |                              | Internationalization Variables for the precedence of internationalization variables                      |  |
| 18808 |                              | used to determine the values of locale categories.)                                                      |  |
| 18809 | <i>LC_ALL</i>                | If set to a non-empty string value, override the values of all the other                                 |  |
| 18810 |                              | internationalization variables.                                                                          |  |
| 18811 | <i>LC_CTYPE</i>              | Determine the locale for the interpretation of sequences of bytes of text data as                        |  |
| 18812 |                              | characters (for example, single-byte as opposed to multi-byte characters in                              |  |
| 18813 |                              | arguments).                                                                                              |  |
| 18814 | <i>LC_MESSAGES</i>           |                                                                                                          |  |
| 18815 |                              | Determine the locale that should be used to affect the format and contents of                            |  |
| 18816 |                              | diagnostic messages written to standard error.                                                           |  |
| 18817 | XSI <i>NLSPATH</i>           | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                    |  |
| 18818 | <b>ASYNCHRONOUS EVENTS</b>   |                                                                                                          |  |
| 18819 |                              | Default.                                                                                                 |  |
| 18820 | <b>STDOUT</b>                |                                                                                                          |  |
| 18821 |                              | If the specified variable is defined on the system and its value is described to be available from       |  |
| 18822 |                              | the <i>confstr()</i> function defined in the System Interfaces volume of IEEE Std 1003.1-2001, its value |  |
| 18823 |                              | shall be written in the following format:                                                                |  |
| 18824 |                              | "%s\n", <value>                                                                                          |  |
| 18825 |                              | Otherwise, if the specified variable is defined on the system, its value shall be written in the         |  |
| 18826 |                              | following format:                                                                                        |  |

```

18827 "%d\n", <value>
18828 If the specified variable is valid, but is undefined on the system, getconf shall write using the
18829 following format:
18830 "undefined\n"
18831 If the variable name is invalid or an error occurs, nothing shall be written to standard output.
18832 STDERR
18833 The standard error shall be used only for diagnostic messages.
18834 OUTPUT FILES
18835 None.
18836 EXTENDED DESCRIPTION
18837 None.
18838 EXIT STATUS
18839 The following exit values shall be returned:
18840 0 The specified variable is valid and information about its current state was written
18841 successfully.
18842 >0 An error occurred.
18843 CONSEQUENCES OF ERRORS
18844 Default.
18845 APPLICATION USAGE
18846 None.
18847 EXAMPLES
18848 The following example illustrates the value of {NGROUPS_MAX}:
18849 getconf NGROUPS_MAX
18850 The following example illustrates the value of {NAME_MAX} for a specific directory:
18851 getconf NAME_MAX /usr
18852 The following example shows how to deal more carefully with results that might be unspecified:
18853 if value=$(getconf PATH_MAX /usr); then
18854 if ["$value" = "undefined"]; then
18855 echo PATH_MAX in /usr is infinite.
18856 else
18857 echo PATH_MAX in /usr is $value.
18858 fi
18859 else
18860 echo Error in getconf.
18861 fi
18862 Note that:
18863 sysconf(_SC_POSIX_C_BIND) ;
18864 and:
18865 system("getconf POSIX2_C_BIND") ;
18866 in a C program could give different answers. The sysconf() call supplies a value that corresponds
18867 to the conditions when the program was either compiled or executed, depending on the

```

18868 implementation; the *system()* call to *getconf* always supplies a value corresponding to conditions  
 18869 when the program is executed.

#### 18870 RATIONALE

18871 The original need for this utility, and for the *confstr()* function, was to provide a way of finding  
 18872 the configuration-defined default value for the *PATH* environment variable. Since *PATH* can be  
 18873 modified by the user to include directories that could contain utilities replacing the standard  
 18874 utilities, shell scripts need a way to determine the system-supplied *PATH* environment variable  
 18875 value that contains the correct search path for the standard utilities. It was later suggested that  
 18876 access to the other variables described in this volume of IEEE Std 1003.1-2001 could also be  
 18877 useful to applications.

18878 This functionality of *getconf* would not be adequately subsumed by another command such as:

18879 `grep var /etc/conf`

18880 because such a strategy would provide correct values for neither those variables that can vary at  
 18881 runtime, nor those that can vary depending on the path.

18882 Early proposal versions of *getconf* specified exit status 1 when the specified variable was valid,  
 18883 but not defined on the system. The output string "undefined" is now used to specify this case  
 18884 with exit code 0 because so many things depend on an exit code of zero when an invoked utility  
 18885 is successful.

#### 18886 FUTURE DIRECTIONS

18887 None.

#### 18888 SEE ALSO

18889 *c99*, the Base Definitions volume of IEEE Std 1003.1-2001, <**limits.h**>, the System Interfaces |  
 18890 volume of IEEE Std 1003.1-2001, *confstr()*, *pathconf()*, *sysconf()*, *system()*

#### 18891 CHANGE HISTORY

18892 First released in Issue 4.

#### 18893 Issue 5

18894 In the OPERANDS section:

- 18895 • {NL\_MAX} is changed to {NL\_NMAX}.
- 18896 • Entries beginning NL\_ are deleted from the list of standard configuration variables.
- 18897 • The list of variables previously marked UX is merged with the list marked EX.
- 18898 • Operands are added to support new Option Groups.
- 18899 • Operands are added so that *getconf* can determine supported programming environments.

#### 18900 Issue 6

18901 The Open Group Corrigendum U029/4 is applied, correcting the example command in the last  
 18902 paragraph of the OPTIONS section.

18903 The following new requirements on POSIX implementations derive from alignment with the  
 18904 Single UNIX Specification:

- 18905 • Operands are added to determine supported programming environments.

18906 This reference page is updated for alignment with the ISO/IEC 9899:1999 standard. Specifically,  
 18907 new macros for *c99* programming environments are introduced.

18908 XSI marked *system\_var* (XBS5\_\*) values are marked LEGACY.

|       |                                                                                               |  |
|-------|-----------------------------------------------------------------------------------------------|--|
| 18909 | IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/27 is applied, correcting the descriptions |  |
| 18910 | of <i>path_var</i> and <i>system_var</i> in the OPERANDS section.                             |  |

18911 **NAME**

18912       getopts — parse utility options

18913 **SYNOPSIS**18914       getopts *optstring name* [*arg...*]18915 **DESCRIPTION**

18916       The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall  
 18917       support the Utility Syntax Guidelines 3 to 10, inclusive, described in the Base Definitions volume  
 18918       of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines.

18919       Each time it is invoked, the *getopts* utility shall place the value of the next option in the shell  
 18920       variable specified by the *name* operand and the index of the next argument to be processed in the  
 18921       shell variable *OPTIND*. Whenever the shell is invoked, *OPTIND* shall be initialized to 1.

18922       When the option requires an option-argument, the *getopts* utility shall place it in the shell  
 18923       variable *OPTARG*. If no option was found, or if the option that was found does not have an  
 18924       option-argument, *OPTARG* shall be unset.

18925       If an option character not contained in the *optstring* operand is found where an option character  
 18926       is expected, the shell variable specified by *name* shall be set to the question-mark ('?') character.  
 18927       In this case, if the first character in *optstring* is a colon (':'), the shell variable *OPTARG* shall be  
 18928       set to the option character found, but no output shall be written to standard error; otherwise, the  
 18929       shell variable *OPTARG* shall be unset and a diagnostic message shall be written to standard  
 18930       error. This condition shall be considered to be an error detected in the way arguments were  
 18931       presented to the invoking application, but shall not be an error in *getopts* processing.

18932       If an option-argument is missing:

- 18933       • If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set to  
 18934       the colon character and the shell variable *OPTARG* shall be set to the option character found.
- 18935       • Otherwise, the shell variable specified by *name* shall be set to the question-mark character,  
 18936       the shell variable *OPTARG* shall be unset, and a diagnostic message shall be written to  
 18937       standard error. This condition shall be considered to be an error detected in the way  
 18938       arguments were presented to the invoking application, but shall not be an error in *getopts*  
 18939       processing; a diagnostic message shall be written as stated, but the exit status shall be zero.

18940       When the end of options is encountered, the *getopts* utility shall exit with a return value greater  
 18941       than zero; the shell variable *OPTIND* shall be set to the index of the first non-option-argument,  
 18942       where the first "--" argument is considered to be an option-argument if there are no other  
 18943       non-option-arguments appearing before it, or the value "\$#+1" if there are no non-option-  
 18944       arguments; the *name* variable shall be set to the question-mark character. Any of the following  
 18945       shall identify the end of options: the special option "--", finding an argument that does not  
 18946       begin with a '-', or encountering an error.

18947       The shell variables *OPTIND* and *OPTARG* shall be local to the caller of *getopts* and shall not be  
 18948       exported by default.

18949       The shell variable specified by the *name* operand, *OPTIND*, and *OPTARG* shall affect the current  
 18950       shell execution environment; see Section 2.12 (on page 61).

18951       If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the  
 18952       current positional parameters or new *arg* values. Any other attempt to invoke *getopts* multiple  
 18953       times in a single shell execution environment with parameters (positional parameters or *arg*  
 18954       operands) that are not the same in all invocations, or with an *OPTIND* value modified to be a  
 18955       value other than 1, produces unspecified results.

18956 **OPTIONS**

18957       None.

18958 **OPERANDS**

18959       The following operands shall be supported:

18960       *optstring*     A string containing the option characters recognized by the utility invoking *getopts*.  
 18961                       If a character is followed by a colon, the option shall be expected to have an  
 18962                       argument, which should be supplied as a separate argument. Applications should  
 18963                       specify an option character and its option-argument as separate arguments, but  
 18964                       *getopts* shall interpret the characters following an option character requiring  
 18965                       arguments as an argument whether or not this is done. An explicit null option-  
 18966                       argument need not be recognized if it is not supplied as a separate argument when  
 18967                       *getopts* is invoked. (See also the *getopt()* function defined in the System Interfaces  
 18968                       volume of IEEE Std 1003.1-2001.) The characters question-mark and colon shall not  
 18969                       be used as option characters by an application. The use of other option characters  
 18970                       that are not alphanumeric produces unspecified results. If the option-argument is  
 18971                       not supplied as a separate argument from the option character, the value in  
 18972                       *OPTARG* shall be stripped of the option character and the *'-'*. The first character  
 18973                       in *optstring* determines how *getopts* behaves if an option character is not known or  
 18974                       an option-argument is missing.

18975       *name*         The name of a shell variable that shall be set by the *getopts* utility to the option  
 18976                       character that was found.

18977       The *getopts* utility by default shall parse positional parameters passed to the invoking shell  
 18978       procedure. If *args* are given, they shall be parsed instead of the positional parameters.

18979 **STDIN**

18980       Not used.

18981 **INPUT FILES**

18982       None.

18983 **ENVIRONMENT VARIABLES**18984       The following environment variables shall affect the execution of *getopts*:

18985       *LANG*         Provide a default value for the internationalization variables that are unset or null.  
 18986                       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 18987                       Internationalization Variables for the precedence of internationalization variables  
 18988                       used to determine the values of locale categories.)

18989       *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 18990                       internationalization variables.

18991       *LC\_CTYPE*     Determine the locale for the interpretation of sequences of bytes of text data as  
 18992                       characters (for example, single-byte as opposed to multi-byte characters in  
 18993                       arguments and input files).

18994       *LC\_MESSAGES*

18995                       Determine the locale that should be used to affect the format and contents of  
 18996                       diagnostic messages written to standard error.

18997 XSI     *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

18998       *OPTIND*       This variable shall be used by the *getopts* utility as the index of the next argument  
 18999                       to be processed.

19000 **ASYNCHRONOUS EVENTS**

19001 Default.

19002 **STDOUT**

19003 Not used.

19004 **STDERR**

19005 Whenever an error is detected and the first character in the *optstring* operand is not a colon  
 19006 (' : '), a diagnostic message shall be written to standard error with the following information in  
 19007 an unspecified format:

19008 • The invoking program name shall be identified in the message. The invoking program name  
 19009 shall be the value of the shell special parameter 0 (see Section 2.5.2 (on page 34)) at the time  
 19010 the *getopts* utility is invoked. A name equivalent to:

19011 `basename "$0"`

19012 may be used.

19013 • If an option is found that was not specified in *optstring*, this error is identified and the invalid  
 19014 option character shall be identified in the message.

19015 • If an option requiring an option-argument is found, but an option-argument is not found,  
 19016 this error shall be identified and the invalid option character shall be identified in the  
 19017 message.

19018 **OUTPUT FILES**

19019 None.

19020 **EXTENDED DESCRIPTION**

19021 None.

19022 **EXIT STATUS**

19023 The following exit values shall be returned:

19024 `0` An option, specified or unspecified by *optstring*, was found.19025 `>0` The end of options was encountered or an error occurred.19026 **CONSEQUENCES OF ERRORS**

19027 Default.

19028 **APPLICATION USAGE**

19029 Since *getopts* affects the current shell execution environment, it is generally provided as a shell  
 19030 regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
 19031 of the following:

19032 `(getopts abc value "$@")`19033 `nohup getopts ...`19034 `find . -exec getopts ... \;`

19035 it does not affect the shell variables in the caller's environment.

19036 Note that shell functions share *OPTIND* with the calling shell even though the positional  
 19037 parameters are changed. If the calling shell and any of its functions uses *getopts* to parse  
 19038 arguments, the results are unspecified.

19039 **EXAMPLES**

19040 The following example script parses and displays its arguments:

19041 `aflag=`19042 `bflag=`



```

19043 while getopts ab: name
19044 do
19045 case $name in
19046 a) aflag=1;;
19047 b) bflag=1
19048 bval="$OPTARG";;
19049 ?) printf "Usage: %s: [-a] [-b value] args\n" $0
19050 exit 2;;
19051 esac
19052 done
19053 if [! -z "$aflag"]; then
19054 printf "Option -a specified\n"
19055 fi
19056 if [! -z "$bflag"]; then
19057 printf 'Option -b "%s" specified\n' "$bval"
19058 fi
19059 shift $(($OPTIND - 1))
19060 printf "Remaining arguments are: %s\n" "$*"

```

#### 19061 RATIONALE

19062 The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles  
 19063 option-arguments containing <blank>s.

19064 The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it  
 19065 does not affect the execution of *getopts*; it is one of the few “output-only” variables used by the  
 19066 standard utilities.

19067 The colon is not allowed as an option character because that is not historical behavior, and it  
 19068 violates the Utility Syntax Guidelines. The colon is now specified to behave as in the KornShell  
 19069 version of the *getopts* utility; when used as the first character in the *optstring* operand, it disables  
 19070 diagnostics concerning missing option-arguments and unexpected option characters. This  
 19071 replaces the use of the *OPTERR* variable that was specified in an early proposal.

19072 The formats of the diagnostic messages produced by the *getopts* utility and the *getopt()* function  
 19073 are not fully specified because implementations with superior (“friendlier”) formats objected to  
 19074 the formats used by some historical implementations. The standard developers considered it  
 19075 important that the information in the messages used be uniform between *getopts* and *getopt()*.  
 19076 Exact duplication of the messages might not be possible, particularly if a utility is built on  
 19077 another system that has a different *getopt()* function, but the messages must have specific  
 19078 information included so that the program name, invalid option character, and type of error can  
 19079 be distinguished by a user.

19080 Only a rare application program intercepts a *getopts* standard error message and wants to parse  
 19081 it. Therefore, implementations are free to choose the most usable messages they can devise. The  
 19082 following formats are used by many historical implementations:

```

19083 "%s: illegal option -- %c\n", <program name>, <option character>
19084 "%s: option requires an argument -- %c\n", <program name>, \
19085 <option character>

```

19086 Historical shells with built-in versions of *getopt()* or *getopts* have used different formats,  
 19087 frequently not even indicating the option character found in error.

19088 **FUTURE DIRECTIONS**

19089           None.

19090 **SEE ALSO**19091           Section 2.5.2 (on page 34), the System Interfaces volume of IEEE Std 1003.1-2001, *getopt()*19092 **CHANGE HISTORY**

19093           First released in Issue 4.

19094 **Issue 6**

19095           The normative text is reworded to avoid use of the term “must” for application requirements.

19096 **NAME**19097 `grep` — search a file for a pattern19098 **SYNOPSIS**19099 `grep [-E | -F] [-c | -l | -q] [-insvx] -e pattern_list...`  
19100 `[-f pattern_file] ... [file...]`19101 `grep [-E | -F] [-c | -l | -q] [-insvx] [-e pattern_list]...`  
19102 `-f pattern_file ... [file...]`19103 `grep [-E | -F] [-c | -l | -q] [-insvx] pattern_list [file...]`19104 **DESCRIPTION**

19105 The *grep* utility shall search the input files, selecting lines matching one or more patterns; the  
 19106 types of patterns are controlled by the options specified. The patterns are specified by the **-e**  
 19107 option, **-f** option, or the *pattern\_list* operand. The *pattern\_list*'s value shall consist of one or more  
 19108 patterns separated by <newline>s; the *pattern\_file*'s contents shall consist of one or more  
 19109 patterns terminated by <newline>. By default, an input line shall be selected if any pattern,  
 19110 treated as an entire basic regular expression (BRE) as described in the Base Definitions volume of  
 19111 IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions, matches any part of the line  
 19112 excluding the terminating <newline>; a null BRE shall match every line. By default, each selected  
 19113 input line shall be written to the standard output.

19114 Regular expression matching shall be based on text lines. Since a <newline> separates or  
 19115 terminates patterns (see the **-e** and **-f** options below), regular expressions cannot contain a  
 19116 <newline>. Similarly, since patterns are matched against individual lines (excluding the  
 19117 terminating <newline>s) of the input, there is no way for a pattern to match a <newline> found  
 19118 in the input.

19119 **OPTIONS**

19120 The *grep* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 19121 12.2, Utility Syntax Guidelines.

19122 The following options shall be supported:

19123 **-E** Match using extended regular expressions. Treat each pattern specified as an ERE,  
 19124 as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.4,  
 19125 Extended Regular Expressions. If any entire ERE pattern matches some part of an  
 19126 input line excluding the terminating <newline>, the line shall be matched. A null  
 19127 ERE shall match every line.

19128 **-F** Match using fixed strings. Treat each pattern specified as a string instead of a  
 19129 regular expression. If an input line contains any of the patterns as a contiguous  
 19130 sequence of bytes, the line shall be matched. A null string shall match every line.

19131 **-c** Write only a count of selected lines to standard output.

19132 **-e *pattern\_list***

19133 Specify one or more patterns to be used during the search for input. The  
 19134 application shall ensure that patterns in *pattern\_list* are separated by a <newline>.  
 19135 A null pattern can be specified by two adjacent <newline>s in *pattern\_list*. Unless  
 19136 the **-E** or **-F** option is also specified, each pattern shall be treated as a BRE, as  
 19137 described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3, Basic  
 19138 Regular Expressions. Multiple **-e** and **-f** options shall be accepted by the *grep*  
 19139 utility. All of the specified patterns shall be used when matching lines, but the  
 19140 order of evaluation is unspecified.

19141       **-f** *pattern\_file*  
 19142           Read one or more patterns from the file named by the pathname *pattern\_file*.  
 19143           Patterns in *pattern\_file* shall be terminated by a <newline>. A null pattern can be  
 19144           specified by an empty line in *pattern\_file*. Unless the **-E** or **-F** option is also  
 19145           specified, each pattern shall be treated as a BRE, as described in the Base  
 19146           Definitions volume of IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions.

19147       **-i**           Perform pattern matching in searches without regard to case; see the Base  
 19148           Definitions volume of IEEE Std 1003.1-2001, Section 9.2, Regular Expression  
 19149           General Requirements.

19150       **-l**           (The letter ell.) Write only the names of files containing selected lines to standard  
 19151           output. Pathnames shall be written once per file searched. If the standard input is  
 19152           searched, a pathname of "(standard input)" shall be written, in the POSIX  
 19153           locale. In other locales, "standard input" may be replaced by something more  
 19154           appropriate in those locales.

19155       **-n**           Precede each output line by its relative line number in the file, each file starting at  
 19156           line 1. The line number counter shall be reset for each file processed.

19157       **-q**           Quiet. Nothing shall be written to the standard output, regardless of matching  
 19158           lines. Exit with zero status if an input line is selected.

19159       **-s**           Suppress the error messages ordinarily written for nonexistent or unreadable files.  
 19160           Other error messages shall not be suppressed.

19161       **-v**           Select lines not matching any of the specified patterns. If the **-v** option is not  
 19162           specified, selected lines shall be those that match any of the specified patterns.

19163       **-x**           Consider only input lines that use all characters in the line excluding the  
 19164           terminating <newline> to match an entire fixed string or regular expression to be  
 19165           matching lines.

## 19166 OPERANDS

19167       The following operands shall be supported:

19168       *pattern\_list*   Specify one or more patterns to be used during the search for input. This operand  
 19169           shall be treated as if it were specified as **-e** *pattern\_list*.

19170       *file*           A pathname of a file to be searched for the patterns. If no *file* operands are  
 19171           specified, the standard input shall be used.

## 19172 STDIN

19173       The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 19174       section.

## 19175 INPUT FILES

19176       The input files shall be text files.

## 19177 ENVIRONMENT VARIABLES

19178       The following environment variables shall affect the execution of *grep*:

19179       *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 19180           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 19181           Internationalization Variables for the precedence of internationalization variables  
 19182           used to determine the values of locale categories.)

19183       *LC\_ALL*          If set to a non-empty string value, override the values of all the other  
 19184           internationalization variables.

19185 **LC\_COLLATE**  
 19186 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 19187 character collating elements within regular expressions.

19188 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 19189 characters (for example, single-byte as opposed to multi-byte characters in  
 19190 arguments and input files) and the behavior of character classes within regular  
 19191 expressions.

19192 **LC\_MESSAGES**  
 19193 Determine the locale that should be used to affect the format and contents of  
 19194 diagnostic messages written to standard error.

19195 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

19196 **ASYNCHRONOUS EVENTS**  
 19197 Default.

19198 **STDOUT**  
 19199 If the **-l** option is in effect, and the **-q** option is not, the following shall be written for each file  
 19200 containing at least one selected input line:  
 19201 "%s\n", <file>  
 19202 Otherwise, if more than one *file* argument appears, and **-q** is not specified, the *grep* utility shall  
 19203 prefix each output line by:  
 19204 "%s: ", <file>  
 19205 The remainder of each output line shall depend on the other options specified:  
 19206 • If the **-c** option is in effect, the remainder of each output line shall contain:  
 19207 "%d\n", <count>  
 19208 • Otherwise, if **-c** is not in effect and the **-n** option is in effect, the following shall be written to  
 19209 standard output:  
 19210 "%d: ", <line number>  
 19211 • Finally, the following shall be written to standard output:  
 19212 "%s", <selected-line contents>

19213 **STDERR**  
 19214 The standard error shall be used only for diagnostic messages.

19215 **OUTPUT FILES**  
 19216 None.

19217 **EXTENDED DESCRIPTION**  
 19218 None.

19219 **EXIT STATUS**  
 19220 The following exit values shall be returned:  
 19221 0 One or more lines were selected.  
 19222 1 No lines were selected.  
 19223 >1 An error occurred.

19224 **CONSEQUENCES OF ERRORS**

19225 If the **-q** option is specified, the exit status shall be zero if an input line is selected, even if an  
 19226 error was detected. Otherwise, default actions shall be performed.

19227 **APPLICATION USAGE**

19228 Care should be taken when using characters in *pattern\_list* that may also be meaningful to the  
 19229 command interpreter. It is safest to enclose the entire *pattern\_list* argument in single quotes:

19230 '...'

19231 The **-e** *pattern\_list* option has the same effect as the *pattern\_list* operand, but is useful when  
 19232 *pattern\_list* begins with the hyphen delimiter. It is also useful when it is more convenient to  
 19233 provide multiple patterns as separate arguments.

19234 Multiple **-e** and **-f** options are accepted and *grep* uses all of the patterns it is given while  
 19235 matching input text lines. (Note that the order of evaluation is not specified. If an  
 19236 implementation finds a null string as a pattern, it is allowed to use that pattern first, matching  
 19237 every line, and effectively ignore any other patterns.)

19238 The **-q** option provides a means of easily determining whether or not a pattern (or string) exists  
 19239 in a group of files. When searching several files, it provides a performance improvement  
 19240 (because it can quit as soon as it finds the first match) and requires less care by the user in  
 19241 choosing the set of files to supply as arguments (because it exits zero if it finds a match even if  
 19242 *grep* detected an access or read error on earlier *file* operands).

19243 **EXAMPLES**

19244 1. To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line  
 19245 numbers:

19246 `grep -i -n posix text.mm`

19247 2. To find all empty lines in the standard input:

19248 `grep ^$`

19249 or:

19250 `grep -v .`

19251 3. Both of the following commands print all lines containing strings "abc" or "def" or both:

19252 `grep -E 'abc|def'`

19253 `grep -F 'abc`

19254 `def'`

19255 4. Both of the following commands print all lines matching exactly "abc" or "def":

19256 `grep -E '^abc$|^def$'`

19257 `grep -F -x 'abc`

19258 `def'`

19259 **RATIONALE**

19260 This *grep* has been enhanced in an upwards-compatible way to provide the exact functionality of  
 19261 the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard  
 19262 developers to consolidate the three *greps* into a single command.

19263 The old *egrep* and *fgrep* commands are likely to be supported for many years to come as  
 19264 implementation extensions, allowing historical applications to operate unmodified.

- 19265 Historical implementations usually silently ignored all but one of multiply-specified **-e** and **-f**  
19266 options, but were not consistent as to which specification was actually used.
- 19267 The **-b** option was omitted from the OPTIONS section because block numbers are  
19268 implementation-defined.
- 19269 The System V restriction on using **-** to mean standard input was omitted.
- 19270 A definition of action taken when given a null BRE or ERE is specified. This is an error condition  
19271 in some historical implementations.
- 19272 The **-I** option previously indicated that its use was undefined when no files were explicitly  
19273 named. This behavior was historical and placed an unnecessary restriction on future  
19274 implementations. It has been removed.
- 19275 The historical BSD *grep* **-s** option practice is easily duplicated by redirecting standard output to  
19276 **/dev/null**. The **-s** option required here is from System V.
- 19277 The **-x** option, historically available only with *fgrep*, is available here for all of the non-  
19278 obsolescent versions.
- 19279 **FUTURE DIRECTIONS**
- 19280 None.
- 19281 **SEE ALSO**
- 19282 *sed*
- 19283 **CHANGE HISTORY**
- 19284 First released in Issue 2.
- 19285 **Issue 6**
- 19286 The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.
- 19287 The normative text is reworded to avoid use of the term “must” for application requirements.
- 19288 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/28 is applied, correcting the examples |  
19289 using the *grep* **-F** option which did not match the normative description of the **-F** option. |

19290 **NAME**

19291 hash — remember or report utility locations

19292 **SYNOPSIS**19293 xSI hash [*utility...*]

19294 hash -r

19295

19296 **DESCRIPTION**

19297 The *hash* utility shall affect the way the current shell environment remembers the locations of  
 19298 utilities found as described in Section 2.9.1.1 (on page 48). Depending on the arguments  
 19299 specified, it shall add utility locations to its list of remembered locations or it shall purge the  
 19300 contents of the list. When no arguments are specified, it shall report on the contents of the list.

19301 Utilities provided as built-ins to the shell shall not be reported by *hash*.

19302 **OPTIONS**

19303 The *hash* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 19304 12.2, Utility Syntax Guidelines.

19305 The following option shall be supported:

19306 -r Forget all previously remembered utility locations.

19307 **OPERANDS**

19308 The following operand shall be supported:

19309 *utility* The name of a utility to be searched for and added to the list of remembered  
 19310 locations. If *utility* contains one or more slashes, the results are unspecified.

19311 **STDIN**

19312 Not used.

19313 **INPUT FILES**

19314 None.

19315 **ENVIRONMENT VARIABLES**

19316 The following environment variables shall affect the execution of *hash*:

19317 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 19318 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 19319 Internationalization Variables for the precedence of internationalization variables  
 19320 used to determine the values of locale categories.)

19321 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 19322 internationalization variables.

19323 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 19324 characters (for example, single-byte as opposed to multi-byte characters in  
 19325 arguments).

19326 *LC\_MESSAGES*

19327 Determine the locale that should be used to affect the format and contents of  
 19328 diagnostic messages written to standard error.

19329 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

19330 *PATH* Determine the location of *utility*, as described in the Base Definitions volume of  
 19331 IEEE Std 1003.1-2001, Chapter 8, Environment Variables.



**19332 ASYNCHRONOUS EVENTS**

19333 Default.

**19334 STDOUT**

19335 The standard output of *hash* shall be used when no arguments are specified. Its format is  
19336 unspecified, but includes the pathname of each utility in the list of remembered locations for the  
19337 current shell environment. This list shall consist of those utilities named in previous *hash*  
19338 invocations that have been invoked, and may contain those invoked and found through the  
19339 normal command search process.

**19340 STDERR**

19341 The standard error shall be used only for diagnostic messages.

**19342 OUTPUT FILES**

19343 None.

**19344 EXTENDED DESCRIPTION**

19345 None.

**19346 EXIT STATUS**

19347 The following exit values shall be returned:

19348 0 Successful completion.

19349 >0 An error occurred.

**19350 CONSEQUENCES OF ERRORS**

19351 Default.

**19352 APPLICATION USAGE**

19353 Since *hash* affects the current shell execution environment, it is always provided as a shell  
19354 regular built-in. If it is called in a separate utility execution environment, such as one of the  
19355 following:

19356 `nohup hash -r`  
19357 `find . -type f | xargs hash`

19358 it does not affect the command search process of the caller's environment.

19359 The *hash* utility may be implemented as an alias—for example, *alias -t -*, in which case utilities  
19360 found through normal command search are not listed by the *hash* command.

19361 The effects of *hash -r* can also be achieved portably by resetting the value of *PATH*; in the  
19362 simplest form, this can be:

19363 `PATH="$PATH"`

19364 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a  
19365 performance improvement on a few implementations; normally, the hashing process is included  
19366 by default.

**19367 EXAMPLES**

19368 None.

**19369 RATIONALE**

19370 None.

**19371 FUTURE DIRECTIONS**

19372 None.

19373 **SEE ALSO**

19374           Section 2.9.1.1 (on page 48)

19375 **CHANGE HISTORY**

19376           First released in Issue 2.

19377 **NAME**

19378           head — copy the first part of files

19379 **SYNOPSIS**19380           head [-n *number*] [*file...*]19381 **DESCRIPTION**19382           The *head* utility shall copy its input files to the standard output, ending the output for each file at  
19383           a designated point.19384           Copying shall end at the point in each input file indicated by the **-n *number*** option. The option-  
19385           argument *number* shall be counted in units of lines.19386 **OPTIONS**19387           The *head* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
19388           12.2, Utility Syntax Guidelines.

19389           The following option shall be supported:

19390           **-n *number***   The first *number* lines of each input file shall be copied to standard output. The  
19391           application shall ensure that the *number* option-argument is a positive decimal  
19392           integer.19393           When a file contains less than *number* lines, it shall be copied to standard output in its entirety.  
19394           This shall not be an error.19395           If no options are specified, *head* shall act as if **-n 10** had been specified.19396 **OPERANDS**

19397           The following operand shall be supported:

19398           *file*           A pathname of an input file. If no *file* operands are specified, the standard input  
19399           shall be used.19400 **STDIN**19401           The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
19402           section.19403 **INPUT FILES**

19404           Input files shall be text files, but the line length is not restricted to {LINE\_MAX} bytes.

19405 **ENVIRONMENT VARIABLES**19406           The following environment variables shall affect the execution of *head*:19407           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
19408           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
19409           Internationalization Variables for the precedence of internationalization variables  
19410           used to determine the values of locale categories.)19411           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
19412           internationalization variables.19413           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
19414           characters (for example, single-byte as opposed to multi-byte characters in  
19415           arguments and input files).19416           **LC\_MESSAGES**19417           Determine the locale that should be used to affect the format and contents of  
19418           diagnostic messages written to standard error.

19419 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 19420 ASYNCHRONOUS EVENTS

19421 Default.

## 19422 STDOUT

19423 The standard output shall contain designated portions of the input files.

19424 If multiple *file* operands are specified, *head* shall precede the output for each with the header:

19425 `"\n==> %s <==\n", <pathname>`

19426 except that the first header written shall not include the initial <newline>.

## 19427 STDERR

19428 The standard error shall be used only for diagnostic messages.

## 19429 OUTPUT FILES

19430 None.

## 19431 EXTENDED DESCRIPTION

19432 None.

## 19433 EXIT STATUS

19434 The following exit values shall be returned:

19435 0 Successful completion.

19436 >0 An error occurred.

## 19437 CONSEQUENCES OF ERRORS

19438 Default.

## 19439 APPLICATION USAGE

19440 The obsolescent *–number* form is withdrawn in this version. Applications should use the *–n*  
19441 *number* option.

## 19442 EXAMPLES

19443 To write the first ten lines of all files (except those with a leading period) in the directory:

19444 `head *`

## 19445 RATIONALE

19446 Although it is possible to simulate *head* with *sed* 10q for a single file, the standard developers  
19447 decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside  
19448 *tail*.

19449 This standard version of *head* follows the Utility Syntax Guidelines. The *–n* option was added to  
19450 this new interface so that *head* and *tail* would be more logically related.

19451 There is no *–c* option (as there is in *tail*) because it is not historical practice and because other  
19452 utilities in this volume of IEEE Std 1003.1-2001 provide similar functionality.

## 19453 FUTURE DIRECTIONS

19454 None.

## 19455 SEE ALSO

19456 *sed*, *tail*

19457 **CHANGE HISTORY**

19458 First released in Issue 4.

19459 **Issue 6**

19460 The obsolescent **–number** form is withdrawn.

19461 The normative text is reworded to avoid use of the term “must” for application requirements.

19462 The DESCRIPTION is updated to clarify that when a file contains less than the number of lines  
19463 requested, the entire file is copied to standard output.

19464 **NAME**

19465 iconv — codeset conversion

19466 **SYNOPSIS**19467 iconv [-cs] -f *frommap* -t *tomap* [*file* ...] |19468 iconv -f *fromcode* [-cs] [-t *tocode* [*file* ...]] |19469 iconv -t *tocode* [-cs] [-f *fromcode*] [*file* ...] |

19470 iconv -l |

19471 **DESCRIPTION**

19472 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and  
 19473 write the results to standard output.

19474 When the options indicate that charmap files are used to specify the codesets (see **OPTIONS**),  
 19475 the codeset conversion shall be accomplished by performing a logical join on the symbolic  
 19476 character names in the two charmaps. The implementation need not support the use of charmap  
 19477 files for codeset conversion unless the POSIX2\_LOCALEDEF symbol is defined on the system.

19478 **OPTIONS**

19479 The *iconv* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 19480 12.2, Utility Syntax Guidelines.

19481 The following options shall be supported:

19482 **-c** Omit any characters that are invalid in the codeset of the input file from the |  
 19483 output. When **-c** is not used, the results of encountering invalid characters in the |  
 19484 input stream (either those that are not characters in the codeset of the input file or |  
 19485 that have no corresponding character in the codeset of the output file) shall be |  
 19486 specified in the system documentation. The presence or absence of **-c** shall not |  
 19487 affect the exit status of *iconv*. |

19488 **-f** *fromcodeset* |  
 19489 Identify the codeset of the input file. The implementation shall recognize the |  
 19490 following two forms of the *fromcodeset* option-argument: |

19491 *fromcode* The *fromcode* option-argument must not contain a slash character. It |  
 19492 shall be interpreted as the name of one of the codeset descriptions |  
 19493 provided by the implementation in an unspecified format. Valid |  
 19494 values of *fromcode* are implementation-defined. |

19495 *frommap* The *frommap* option-argument must contain a slash character. It shall |  
 19496 be interpreted as the pathname of a charmap file as defined in the |  
 19497 Base Definitions volume of IEEE Std 1003.1-2001, Section 6.4,  
 19498 Character Set Description File. If the pathname does not represent a  
 19499 valid, readable charmap file, the results are undefined. |

19500 If this option is omitted, the codeset of the current locale shall be used. |

19501 **-l** Write all supported *fromcode* and *tocode* values to standard output in an unspecified  
 19502 format.

19503 **-s** Suppress any messages written to standard error concerning invalid characters.  
 19504 When **-s** is not used, the results of encountering invalid characters in the input  
 19505 stream (either those that are not valid characters in the codeset of the input file or  
 19506 that have no corresponding character in the codeset of the output file) shall be  
 19507 specified in the system documentation. The presence or absence of **-s** shall not  
 19508 affect the exit status of *iconv*. |

19509        **-t tocodeset**   Identify the codeset to be used for the output file. The implementation shall  
 19510                            recognize the following two forms of the *tocodeset* option-argument:  
 19511                            *tocode*        The semantics shall be equivalent to the **-f fromcode** option.  
 19512                            *tomap*         The semantics shall be equivalent to the *tomap* option.  
 19513                            If this option is omitted, the codeset of the current locale shall be used.  
 19514        If either **-f** or **-t** represents a charmap file, but the other does not (or is omitted), or both **-f** and  
 19515        **-t** are omitted, the results are undefined.

#### 19516 **OPERANDS**

19517        The following operand shall be supported:

19518        *file*            A pathname of an input file. If no *file* operands are specified, or if a *file* operand is  
 19519                            ' - ', the standard input shall be used.

#### 19520 **STDIN**

19521        The standard input shall be used only if no *file* operands are specified, or if a *file* operand is ' - '.

#### 19522 **INPUT FILES**

19523        The input file shall be a text file.

#### 19524 **ENVIRONMENT VARIABLES**

19525        The following environment variables shall affect the execution of *iconv*:

19526        **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 19527                            (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 19528                            Internationalization Variables for the precedence of internationalization variables  
 19529                            used to determine the values of locale categories.)

19530        **LC\_ALL**         If set to a non-empty string value, override the values of all the other  
 19531                            internationalization variables.

19532        **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
 19533                            characters (for example, single-byte as opposed to multi-byte characters in  
 19534                            arguments). During translation of the file, this variable is superseded by the use of  
 19535                            the *fromcode* option-argument.

19536        **LC\_MESSAGES**     Determine the locale that should be used to affect the format and contents of  
 19537                            diagnostic messages written to standard error.  
 19538

19539 **XSI**        **NLSPATH**     Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

#### 19540 **ASYNCHRONOUS EVENTS**

19541        Default.

#### 19542 **STDOUT**

19543        When the **-I** option is used, the standard output shall contain all supported *fromcode* and *tocode*  
 19544        values, written in an unspecified format.

19545        When the **-I** option is not used, the standard output shall contain the sequence of characters  
 19546        read from the input files, translated to the specified codeset. Nothing else shall be written to the  
 19547        standard output.

#### 19548 **STDERR**

19549        The standard error shall be used only for diagnostic messages.

19550 **OUTPUT FILES**

19551       None.

19552 **EXTENDED DESCRIPTION**

19553       None.

19554 **EXIT STATUS**

19555       The following exit values shall be returned:

19556       0   Successful completion.

19557       &gt;0   An error occurred.

19558 **CONSEQUENCES OF ERRORS**

19559       Default.

19560 **APPLICATION USAGE**

19561       The user must ensure that both charmap files use the same symbolic names for characters the  
 19562       two codesets have in common.

19563 **EXAMPLES**

19564       The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:1994  
 19565       standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file  
 19566       **mail.local**:

19567       iconv -f IS6937 -t IS8859 mail.x400 &gt; mail.local

19568 **RATIONALE**

19569       The *iconv* utility can be used portably only when the user provides two charmap files as option-  
 19570       arguments. This is because a single charmap provided by the user cannot reliably be joined with  
 19571       the names in a system-provided character set description. The valid values for *fromcode* and  
 19572       *toencode* are implementation-defined and do not have to have any relation to the charmap  
 19573       mechanisms. As an aid to interactive users, the **-l** option was adopted from the Plan 9 operating  
 19574       system. It writes information concerning these implementation-defined values. The format is  
 19575       unspecified because there are many possible useful formats that could be chosen, such as a  
 19576       matrix of valid combinations of *fromcode* and *toencode*. The **-l** option is not intended for shell script  
 19577       usage; conforming applications will have to use charmaps.

19578 **FUTURE DIRECTIONS**

19579       None.

19580 **SEE ALSO**19581       *gencat*19582 **CHANGE HISTORY**

19583       First released in Issue 3.

19584 **Issue 6**

19585       This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the  
 19586       ability to use charmap files for conversion has been added.

19587       IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/29 is applied, making changes to address  
 19588       inconsistencies with the *iconv()* function in the System Interfaces volume of  
 19589       IEEE Std 1003.1-2001.



19590 **NAME**

19591           id — return user identity

19592 **SYNOPSIS**19593           id [*user*]19594           id -G[-n] [*user*]19595           id -g[-nr] [*user*]19596           id -u[-nr] [*user*]19597 **DESCRIPTION**

19598           If no *user* operand is provided, the *id* utility shall write the user and group IDs and the  
 19599           corresponding user and group names of the invoking process to standard output. If the effective  
 19600           and real IDs do not match, both shall be written. If multiple groups are supported by the  
 19601           underlying system (see the description of {NGROUPS\_MAX} in the System Interfaces volume of  
 19602           IEEE Std 1003.1-2001), the supplementary group affiliations of the invoking process shall also be  
 19603           written.

19604           If a *user* operand is provided and the process has the appropriate privileges, the user and group  
 19605           IDs of the selected user shall be written. In this case, effective IDs shall be assumed to be  
 19606           identical to real IDs. If the selected user has more than one allowable group membership listed  
 19607           in the group database, these shall be written in the same manner as the supplementary groups  
 19608           described in the preceding paragraph.

19609 **OPTIONS**

19610           The *id* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 19611           Utility Syntax Guidelines.

19612           The following options shall be supported:

19613           **-G**           Output all different group IDs (effective, real, and supplementary) only, using the  
 19614           format "%u\n". If there is more than one distinct group affiliation, output each  
 19615           such affiliation, using the format " %u", before the <newline> is output.

19616           **-g**           Output only the effective group ID, using the format "%u\n".

19617           **-n**           Output the name in the format "%s" instead of the numeric ID using the format  
 19618           "%u".

19619           **-r**           Output the real ID instead of the effective ID.

19620           **-u**           Output only the effective user ID, using the format "%u\n".

19621 **OPERANDS**

19622           The following operand shall be supported:

19623           *user*           The login name for which information is to be written.

19624 **STDIN**

19625           Not used.

19626 **INPUT FILES**

19627           None.

19628 **ENVIRONMENT VARIABLES**

19629           The following environment variables shall affect the execution of *id*:

19630           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
 19631           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 19632           Internationalization Variables for the precedence of internationalization variables

19633                   used to determine the values of locale categories.)

19634       *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
19635                   internationalization variables.

19636       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
19637                   characters (for example, single-byte as opposed to multi-byte characters in  
19638                   arguments).

19639       *LC\_MESSAGES*  
19640                   Determine the locale that should be used to affect the format and contents of  
19641                   diagnostic messages written to standard error and informative messages written to  
19642                   standard output.

19643 XSI       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

19644 **ASYNCHRONOUS EVENTS**

19645       Default.

19646 **STDOUT**

19647       The following formats shall be used when the *LC\_MESSAGES* locale category specifies the  
19648       POSIX locale. In other locales, the strings *uid*, *gid*, *euid*, *egid*, and *groups* may be replaced with  
19649       more appropriate strings corresponding to the locale.

19650       "uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,  
19651        <real group ID>, <group-name>

19652       If the effective and real user IDs do not match, the following shall be inserted immediately  
19653       before the '\n' character in the previous format:

19654       " euid=%u(%s) "

19655       with the following arguments added at the end of the argument list:

19656       <effective user ID>, <effective user-name>

19657       If the effective and real group IDs do not match, the following shall be inserted directly before  
19658       the '\n' character in the format string (and after any addition resulting from the effective and  
19659       real user IDs not matching):

19660       " egid=%u(%s) "

19661       with the following arguments added at the end of the argument list:

19662       <effective group-ID>, <effective group name>

19663       If the process has supplementary group affiliations or the selected user is allowed to belong to  
19664       multiple groups, the first shall be added directly before the <newline> in the format string:

19665       " groups=%u(%s) "

19666       with the following arguments added at the end of the argument list:

19667       <supplementary group ID>, <supplementary group name>

19668       and the necessary number of the following added after that for any remaining supplementary  
19669       group IDs:

19670       ", %u(%s) "

19671       and the necessary number of the following arguments added at the end of the argument list:

19672       <supplementary group ID>, <supplementary group name>

19673 If any of the user ID, group ID, effective user ID, effective group ID, or supplementary /multiple  
19674 group IDs cannot be mapped by the system into printable user or group names, the  
19675 corresponding " (%s) " and *name* argument shall be omitted from the corresponding format  
19676 string.

19677 When any of the options are specified, the output format shall be as described in the OPTIONS  
19678 section.

#### 19679 **STDERR**

19680 The standard error shall be used only for diagnostic messages.

#### 19681 **OUTPUT FILES**

19682 None.

#### 19683 **EXTENDED DESCRIPTION**

19684 None.

#### 19685 **EXIT STATUS**

19686 The following exit values shall be returned:

19687 0 Successful completion.

19688 >0 An error occurred.

#### 19689 **CONSEQUENCES OF ERRORS**

19690 Default.

#### 19691 **APPLICATION USAGE**

19692 Output produced by the **-G** option and by the default case could potentially produce very long  
19693 lines on systems that support large numbers of supplementary groups. (On systems with user  
19694 and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per  
19695 name, 93 supplementary groups plus distinct effective and real group and user IDs could  
19696 theoretically overflow the 2048-byte {LINE\_MAX} text file line limit on the default output case.  
19697 It would take about 186 supplementary groups to overflow the 2048-byte barrier using *id -G*).  
19698 This is not expected to be a problem in practice, but in cases where it is a concern, applications  
19699 should consider using *fold -s* before postprocessing the output of *id*.

#### 19700 **EXAMPLES**

19701 None.

#### 19702 **RATIONALE**

19703 The functionality provided by the 4 BSD *groups* utility can be simulated using:

19704 `id -Gn [ user ]`

19705 The 4 BSD command *groups* was considered, but it was not included because it did not provide  
19706 the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to  
19707 modify *id* to provide the additional functionality necessary to systems with multiple groups  
19708 than to invent another command.

19709 The options **-u**, **-g**, **-n**, and **-r** were added to ease the use of *id* with shell commands  
19710 substitution. Without these options it is necessary to use some preprocessor such as *sed* to select  
19711 the desired piece of information. Since output such as that produced by:

19712 `id -u -n`

19713 is frequently wanted, it seemed desirable to add the options.

19714 **FUTURE DIRECTIONS**

19715       None.

19716 **SEE ALSO**19717       *fold*, *logname*, *who*, the System Interfaces volume of IEEE Std 1003.1-2001, *getgid()*, *getgroups()*,  
19718       *getuid()*19719 **CHANGE HISTORY**

19720       First released in Issue 2.

19721 **NAME**

19722 ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

19723 **SYNOPSIS**

```
19724 xsi ipcrm [-q msgid | -Q msgkey | -s semid | -S semkey |
19725 -m shmid | -M shmkey] ...
19726
```

19727 **DESCRIPTION**

19728 The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory  
 19729 segments. The interprocess communication facilities to be removed are specified by the options.

19730 Only a user with appropriate privilege shall be allowed to remove an interprocess  
 19731 communication facility that was not created by or owned by the user invoking *ipcrm*.

19732 **OPTIONS**

19733 The *ipcrm* facility supports the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 19734 Utility Syntax Guidelines.

19735 The following options shall be supported:

19736 **-q msgid** Remove the message queue identifier *msgid* from the system and destroy the  
 19737 message queue and data structure associated with it.

19738 **-m shmid** Remove the shared memory identifier *shmid* from the system. The shared memory  
 19739 segment and data structure associated with it shall be destroyed after the last  
 19740 detach.

19741 **-s semid** Remove the semaphore identifier *semid* from the system and destroy the set of  
 19742 semaphores and data structure associated with it.

19743 **-Q msgkey** Remove the message queue identifier, created with key *msgkey*, from the system  
 19744 and destroy the message queue and data structure associated with it.

19745 **-M shmkey** Remove the shared memory identifier, created with key *shmkey*, from the system.  
 19746 The shared memory segment and data structure associated with it shall be  
 19747 destroyed after the last detach.

19748 **-S semkey** Remove the semaphore identifier, created with key *semkey*, from the system and  
 19749 destroy the set of semaphores and data structure associated with it.

19750 **OPERANDS**

19751 None.

19752 **STDIN**

19753 Not used.

19754 **INPUT FILES**

19755 None.

19756 **ENVIRONMENT VARIABLES**

19757 The following environment variables shall affect the execution of *ipcrm*:

19758 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 19759 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 19760 Internationalization Variables for the precedence of internationalization variables  
 19761 used to determine the values of locale categories.)

19762 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 19763 internationalization variables.

19764        *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 19765                           characters (for example, single-byte as opposed to multi-byte characters in  
 19766                           arguments).

19767        *LC\_MESSAGES*  
 19768                           Determine the locale that should be used to affect the format and contents of  
 19769                           diagnostic messages written to standard error.

19770        *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

19771 **ASYNCHRONOUS EVENTS**  
 19772        Default.

19773 **STDOUT**  
 19774        Not used.

19775 **STDERR**  
 19776        The standard error shall be used only for diagnostic messages.

19777 **OUTPUT FILES**  
 19778        None.

19779 **EXTENDED DESCRIPTION**  
 19780        None.

19781 **EXIT STATUS**  
 19782        The following exit values shall be returned:  
 19783        0   Successful completion.  
 19784        >0   An error occurred.

19785 **CONSEQUENCES OF ERRORS**  
 19786        Default.

19787 **APPLICATION USAGE**  
 19788        None.

19789 **EXAMPLES**  
 19790        None.

19791 **RATIONALE**  
 19792        None.

19793 **FUTURE DIRECTIONS**  
 19794        None.

19795 **SEE ALSO**  
 19796        *ipcs*, the System Interfaces volume of IEEE Std 1003.1-2001, *msgctl()*, *semctl()*, *shmctl()*

19797 **CHANGE HISTORY**  
 19798        First released in Issue 5.

19799 **NAME**19800 `ipcs` — report XSI interprocess communication facilities status19801 **SYNOPSIS**19802 xsi `ipcs [-qms] [-a | -bcopt]`

19803

19804 **DESCRIPTION**19805 The *ipcs* utility shall write information about active interprocess communication facilities.

19806 Without options, information shall be written in short format for message queues, shared  
 19807 memory segments, and semaphore sets that are currently active in the system. Otherwise, the  
 19808 information that is displayed is controlled by the options specified.

19809 **OPTIONS**

19810 The *ipcs* facility supports the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 19811 Utility Syntax Guidelines.

19812 The *ipcs* utility accepts the following options:19813 **-q** Write information about active message queues.19814 **-m** Write information about active shared memory segments.19815 **-s** Write information about active semaphore sets.

19816 If **-q**, **-m**, or **-s** are specified, only information about those facilities shall be written. If none of  
 19817 these three are specified, information about all three shall be written subject to the following  
 19818 options:

19819 **-a** Use all print options. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

19820 **-b** Write information on maximum allowable size. (Maximum number of bytes in  
 19821 messages on queue for message queues, size of segments for shared memory, and  
 19822 number of semaphores in each set for semaphores.)

19823 **-c** Write creator's user name and group name; see below.

19824 **-o** Write information on outstanding usage. (Number of messages on queue and total  
 19825 number of bytes in messages on queue for message queues, and number of  
 19826 processes attached to shared memory segments.)

19827 **-p** Write process number information. (Process ID of the last process to send a  
 19828 message and process ID of the last process to receive a message on message  
 19829 queues, process ID of the creating process, and process ID of the last process to  
 19830 attach or detach on shared memory segments.)

19831 **-t** Write time information. (Time of the last control operation that changed the access  
 19832 permissions for all facilities, time of the last *msgsnd()* and *msgrcv()* operations on  
 19833 message queues, time of the last *shmat()* and *shmdt()* operations on shared  
 19834 memory, and time of the last *semop()* operation on semaphores.)

19835 **OPERANDS**

19836 None.

19837 **STDIN**

19838 Not used.

19839 **INPUT FILES**

- 19840           • The group database
- 19841           • The user database

19842 **ENVIRONMENT VARIABLES**

19843           The following environment variables shall affect the execution of *ipcs*:

- 19844           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
19845                       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
19846                       Internationalization Variables for the precedence of internationalization variables  
19847                       used to determine the values of locale categories.)
- 19848           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
19849                       internationalization variables.
- 19850           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
19851                       characters (for example, single-byte as opposed to multi-byte characters in  
19852                       arguments).
- 19853           *LC\_MESSAGES*  
19854                       Determine the locale that should be used to affect the format and contents of  
19855                       diagnostic messages written to standard error.
- 19856           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 19857           *TZ*         Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset  
19858                       or null, an unspecified default timezone shall be used.

19859 **ASYNCHRONOUS EVENTS**

19860           Default.

19861 **STDOUT**

19862           An introductory line shall be written with the format:

19863           "IPC status from %s as of %s\n", <source>, <date>

19864           where <source> indicates the source used to gather the statistics and <date> is the information  
19865           that would be produced by the *date* command when invoked in the POSIX locale.

19866           The *ipcs* utility then shall create up to three reports depending upon the *-q*, *-m*, and *-s* options.  
19867           The first report shall indicate the status of message queues, the second report shall indicate the  
19868           status of shared memory segments, and the third report shall indicate the status of semaphore  
19869           sets.

19870           If the corresponding facility is not installed or has not been used since the last reboot, then the  
19871           report shall be written out in the format:

19872           "%s facility not in system.\n", <facility>

19873           where <facility> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has  
19874           been installed and has been used since the last reboot, column headings separated by one or  
19875           more spaces and followed by a <newline> shall be written as indicated below followed by the  
19876           facility name written out using the format:

19877           "%s:\n", <facility>

19878           where <facility> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second  
19879           and third reports the column headings need not be written if the last column headings written  
19880           already provide column headings for all information in that report.



|       |                                                                                                   |              |                                                                                                     |
|-------|---------------------------------------------------------------------------------------------------|--------------|-----------------------------------------------------------------------------------------------------|
| 19881 | The column headings provided in the first column below and the meaning of the information in      |              |                                                                                                     |
| 19882 | those columns shall be given in order below; the letters in parentheses indicate the options that |              |                                                                                                     |
| 19883 | shall cause the corresponding column to appear; “all” means that the column shall always          |              |                                                                                                     |
| 19884 | appear. Each column is separated by one or more <space>s. Note that these options only            |              |                                                                                                     |
| 19885 | determine what information is provided for each report; they do not determine which reports       |              |                                                                                                     |
| 19886 | are written.                                                                                      |              |                                                                                                     |
| 19887 | T                                                                                                 | (all)        | Type of facility:                                                                                   |
| 19888 |                                                                                                   | q            | Message queue.                                                                                      |
| 19889 |                                                                                                   | m            | Shared memory segment.                                                                              |
| 19890 |                                                                                                   | s            | Semaphore.                                                                                          |
| 19891 |                                                                                                   |              | This field is a single character written using the format %c.                                       |
| 19892 | ID                                                                                                | (all)        | The identifier for the facility entry. This field shall be written using the format                 |
| 19893 |                                                                                                   |              | %d.                                                                                                 |
| 19894 | KEY                                                                                               | (all)        | The key used as an argument to <i>msgget()</i> , <i>semget()</i> , or <i>shmget()</i> to create the |
| 19895 |                                                                                                   |              | facility entry.                                                                                     |
| 19896 |                                                                                                   | <b>Note:</b> | The key of a shared memory segment is changed to IPC_PRIVATE when                                   |
| 19897 |                                                                                                   |              | the segment has been removed until all processes attached to the segment                            |
| 19898 |                                                                                                   |              | detach it.                                                                                          |
| 19899 |                                                                                                   |              | This field shall be written using the format 0x%x.                                                  |
| 19900 | MODE                                                                                              | (all)        | The facility access modes and flags. The mode shall consist of 11 characters                        |
| 19901 |                                                                                                   |              | that are interpreted as follows.                                                                    |
| 19902 |                                                                                                   |              | The first character shall be:                                                                       |
| 19903 |                                                                                                   | S            | If a process is waiting on a <i>msgsnd()</i> operation.                                             |
| 19904 |                                                                                                   | –            | If the above is not true.                                                                           |
| 19905 |                                                                                                   |              | The second character shall be:                                                                      |
| 19906 |                                                                                                   | R            | If a process is waiting on a <i>msgrcv()</i> operation.                                             |
| 19907 |                                                                                                   | C or –       | If the associated shared memory segment is to be cleared when the                                   |
| 19908 |                                                                                                   |              | first attach operation is executed.                                                                 |
| 19909 |                                                                                                   | –            | If none of the above is true.                                                                       |
| 19910 |                                                                                                   |              | The next nine characters shall be interpreted as three sets of three bits each.                     |
| 19911 |                                                                                                   |              | The first set refers to the owner’s permissions; the next to permissions of                         |
| 19912 |                                                                                                   |              | others in the usergroup of the facility entry; and the last to all others. Within                   |
| 19913 |                                                                                                   |              | each set, the first character indicates permission to read, the second character                    |
| 19914 |                                                                                                   |              | indicates permission to write or alter the facility entry, and the last character is                |
| 19915 |                                                                                                   |              | a minus sign (‘–’).                                                                                 |
| 19916 |                                                                                                   |              | The permissions shall be indicated as follows:                                                      |
| 19917 |                                                                                                   | r            | If read permission is granted.                                                                      |
| 19918 |                                                                                                   | w            | If write permission is granted.                                                                     |
| 19919 |                                                                                                   | a            | If alter permission is granted.                                                                     |
| 19920 |                                                                                                   | –            | If the indicated permission is not granted.                                                         |

|       |                                                                          |       |                                                                                   |
|-------|--------------------------------------------------------------------------|-------|-----------------------------------------------------------------------------------|
| 19921 |                                                                          |       | The first character following the permissions specifies if there is an alternate  |
| 19922 |                                                                          |       | or additional access control method associated with the facility. If there is no  |
| 19923 |                                                                          |       | alternate or additional access control method associated with the facility, a     |
| 19924 |                                                                          |       | single <space> shall be written; otherwise, another printable character is        |
| 19925 |                                                                          |       | written.                                                                          |
| 19926 | OWNER                                                                    | (all) | The user name of the owner of the facility entry. If the user name of the owner   |
| 19927 |                                                                          |       | is found in the user database, at least the first eight column positions of the   |
| 19928 |                                                                          |       | name shall be written using the format %s. Otherwise, the user ID of the          |
| 19929 |                                                                          |       | owner shall be written using the format %d.                                       |
| 19930 | GROUP                                                                    | (all) | The group name of the owner of the facility entry. If the group name of the       |
| 19931 |                                                                          |       | owner is found in the group database, at least the first eight column positions   |
| 19932 |                                                                          |       | of the name shall be written using the format %s. Otherwise, the group ID of      |
| 19933 |                                                                          |       | the owner shall be written using the format %d.                                   |
| 19934 | The following nine columns shall be only written out for message queues: |       |                                                                                   |
| 19935 | CREATOR                                                                  | (a,c) | The user name of the creator of the facility entry. If the user name of the       |
| 19936 |                                                                          |       | creator is found in the user database, at least the first eight column positions  |
| 19937 |                                                                          |       | of the name shall be written using the format %s. Otherwise, the user ID of       |
| 19938 |                                                                          |       | the creator shall be written using the format %d.                                 |
| 19939 | CGROUP                                                                   | (a,c) | The group name of the creator of the facility entry. If the group name of the     |
| 19940 |                                                                          |       | creator is found in the group database, at least the first eight column positions |
| 19941 |                                                                          |       | of the name shall be written using the format %s. Otherwise, the group ID of      |
| 19942 |                                                                          |       | the creator shall be written using the format %d.                                 |
| 19943 | CBYTES                                                                   | (a,o) | The number of bytes in messages currently outstanding on the associated           |
| 19944 |                                                                          |       | message queue. This field shall be written using the format %d.                   |
| 19945 | QNUM                                                                     | (a,o) | The number of messages currently outstanding on the associated message            |
| 19946 |                                                                          |       | queue. This field shall be written using the format %d.                           |
| 19947 | QBYTES                                                                   | (a,b) | The maximum number of bytes allowed in messages outstanding on the                |
| 19948 |                                                                          |       | associated message queue. This field shall be written using the format %d.        |
| 19949 | LSPID                                                                    | (a,p) | The process ID of the last process to send a message to the associated queue.     |
| 19950 |                                                                          |       | This field shall be written using the format:                                     |
| 19951 |                                                                          |       | "%d", <pid>                                                                       |
| 19952 |                                                                          |       | where <pid> is 0 if no message has been sent to the corresponding message         |
| 19953 |                                                                          |       | queue; otherwise, <pid> shall be the process ID of the last process to send a     |
| 19954 |                                                                          |       | message to the queue.                                                             |
| 19955 | LRPID                                                                    | (a,p) | The process ID of the last process to receive a message from the associated       |
| 19956 |                                                                          |       | queue. This field shall be written using the format:                              |
| 19957 |                                                                          |       | "%d", <pid>                                                                       |
| 19958 |                                                                          |       | where <pid> is 0 if no message has been received from the corresponding           |
| 19959 |                                                                          |       | message queue; otherwise, <pid> shall be the process ID of the last process to    |
| 19960 |                                                                          |       | receive a message from the queue.                                                 |
| 19961 | STIME                                                                    | (a,t) | The time the last message was sent to the associated queue. If a message has      |
| 19962 |                                                                          |       | been sent to the corresponding message queue, the hour, minute, and second        |
| 19963 |                                                                          |       | of the last time a message was sent to the queue shall be written using the       |
| 19964 |                                                                          |       | format %d:%2.2d:%2.2d. Otherwise, the format "no-entry" shall be                  |
| 19965 |                                                                          |       | written.                                                                          |

|       |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
|-------|-----------------------------------------------------------------------------------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 19966 | RTIME                                                                             | (a,t) | The time the last message was received from the associated queue. If a message has been received from the corresponding message queue, the hour, minute, and second of the last time a message was received from the queue shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written. |
| 19967 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19968 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19969 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19970 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19971 | The following eight columns shall be only written out for shared memory segments. |       |                                                                                                                                                                                                                                                                                                                                  |
| 19972 | CREATOR                                                                           | (a,c) | The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.                                            |
| 19973 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19974 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19975 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19976 | CGROUP                                                                            | (a,c) | The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.                                   |
| 19977 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19978 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19979 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19980 | NATTCH                                                                            | (a,o) | The number of processes attached to the associated shared memory segment. This field shall be written using the format %d.                                                                                                                                                                                                       |
| 19981 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19982 | SEGSZ                                                                             | (a,b) | The size of the associated shared memory segment. This field shall be written using the format %d.                                                                                                                                                                                                                               |
| 19983 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19984 | CPID                                                                              | (a,p) | The process ID of the creator of the shared memory entry. This field shall be written using the format %d.                                                                                                                                                                                                                       |
| 19985 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19986 | LPID                                                                              | (a,p) | The process ID of the last process to attach or detach the shared memory segment. This field shall be written using the format:                                                                                                                                                                                                  |
| 19987 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19988 |                                                                                   |       | "%d", <pid>                                                                                                                                                                                                                                                                                                                      |
| 19989 |                                                                                   |       | where <pid> is 0 if no process has attached the corresponding shared memory segment; otherwise, <pid> shall be the process ID of the last process to attach or detach the segment.                                                                                                                                               |
| 19990 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19991 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19992 | ATIME                                                                             | (a,t) | The time the last attach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been attached, the hour, minute, and second of the last time the segment was attached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.  |
| 19993 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19994 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19995 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19996 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19997 | DTIME                                                                             | (a,t) | The time the last detach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been detached, the hour, minute, and second of the last time the segment was detached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.  |
| 19998 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 19999 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 20000 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 20001 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 20002 | The following four columns shall be only written out for semaphore sets:          |       |                                                                                                                                                                                                                                                                                                                                  |
| 20003 | CREATOR                                                                           | (a,c) | The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.                                            |
| 20004 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 20005 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 20006 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 20007 | CGROUP                                                                            | (a,c) | The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.                                   |
| 20008 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 20009 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |
| 20010 |                                                                                   |       |                                                                                                                                                                                                                                                                                                                                  |

|       |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
|-------|-------------------------------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20011 | NSEMS                         | (a,b) | The number of semaphores in the set associated with the semaphore entry.                                                                                                                                                                                                                                                                                                         |
| 20012 |                               |       | This field shall be written using the format %d.                                                                                                                                                                                                                                                                                                                                 |
| 20013 | OTIME                         | (a,t) | The time the last semaphore operation on the set associated with the semaphore entry was completed. If a semaphore operation has ever been performed on the corresponding semaphore set, the hour, minute, and second of the last semaphore operation on the semaphore set shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written. |
| 20014 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20015 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20016 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20017 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20018 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20019 |                               |       | The following column shall be written for all three reports when it is requested:                                                                                                                                                                                                                                                                                                |
| 20020 | CTIME                         | (a,t) | The time the associated entry was created or changed. The hour, minute, and second of the time when the associated entry was created shall be written using the format %d:%2.2d:%2.2d.                                                                                                                                                                                           |
| 20021 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20022 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20023 | <b>STDERR</b>                 |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20024 |                               |       | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                                                                                                                                   |
| 20025 | <b>OUTPUT FILES</b>           |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20026 |                               |       | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 20027 | <b>EXTENDED DESCRIPTION</b>   |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20028 |                               |       | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 20029 | <b>EXIT STATUS</b>            |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20030 |                               |       | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                     |
| 20031 | 0                             |       | Successful completion.                                                                                                                                                                                                                                                                                                                                                           |
| 20032 | >0                            |       | An error occurred.                                                                                                                                                                                                                                                                                                                                                               |
| 20033 | <b>CONSEQUENCES OF ERRORS</b> |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20034 |                               |       | Default.                                                                                                                                                                                                                                                                                                                                                                         |
| 20035 | <b>APPLICATION USAGE</b>      |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20036 |                               |       | Things can change while <i>ipcs</i> is running; the information it gives is guaranteed to be accurate only when it was retrieved.                                                                                                                                                                                                                                                |
| 20037 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20038 | <b>EXAMPLES</b>               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20039 |                               |       | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 20040 | <b>RATIONALE</b>              |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20041 |                               |       | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 20042 | <b>FUTURE DIRECTIONS</b>      |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20043 |                               |       | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 20044 | <b>SEE ALSO</b>               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20045 |                               |       | The System Interfaces volume of IEEE Std 1003.1-2001, <i>msgrcv()</i> , <i>msgsnd()</i> , <i>semget()</i> , <i>semop()</i> , <i>shmat()</i> , <i>shmdt()</i> , <i>shmget()</i>                                                                                                                                                                                                   |
| 20046 |                               |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20047 | <b>CHANGE HISTORY</b>         |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20048 |                               |       | First released in Issue 5.                                                                                                                                                                                                                                                                                                                                                       |
| 20049 | <b>Issue 6</b>                |       |                                                                                                                                                                                                                                                                                                                                                                                  |
| 20050 |                               |       | The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.                                                                                                                                                                                                                                                                                                           |
| 20051 |                               |       | The Open Group Corrigenda U032/1 and U032/2 are applied, clarifying the output format.                                                                                                                                                                                                                                                                                           |
| 20052 |                               |       | The Open Group Base Resolution bwg98-004 is applied.                                                                                                                                                                                                                                                                                                                             |

20053 **NAME**20054           *jobs* — display status of jobs in the current session20055 **SYNOPSIS**20056 UP       *jobs* [-l | -p] [*job\_id*...]

20057

20058 **DESCRIPTION**20059           The *jobs* utility shall display the status of jobs that were started in the current shell environment;  
20060           see Section 2.12 (on page 61).20061           When *jobs* reports the termination status of a job, the shell shall remove its process ID from the  
20062           list of those “known in the current shell execution environment”; see Section 2.9.3.1 (on page  
20063           50).20064 **OPTIONS**20065           The *jobs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
20066           12.2, Utility Syntax Guidelines.

20067           The following options shall be supported:

20068           **-l**           (The letter ell.) Provide more information about each job listed. This information  
20069                           shall include the job number, current job, process group ID, state, and the  
20070                           command that formed the job.20071           **-p**           Display only the process IDs for the process group leaders of the selected jobs.20072           By default, the *jobs* utility shall display the status of all stopped jobs, running background jobs  
20073           and all jobs whose status has changed and have not been reported by the shell.20074 **OPERANDS**

20075           The following operand shall be supported:

20076           *job\_id*       Specifies the jobs for which the status is to be displayed. If no *job\_id* is given, the  
20077                           status information for all jobs shall be displayed. The format of *job\_id* is described  
20078                           in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.203, Job Control  
20079                           Job ID.20080 **STDIN**

20081           Not used.

20082 **INPUT FILES**

20083           None.

20084 **ENVIRONMENT VARIABLES**20085           The following environment variables shall affect the execution of *jobs*:20086           **LANG**       Provide a default value for the internationalization variables that are unset or null.  
20087                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
20088                           Internationalization Variables for the precedence of internationalization variables  
20089                           used to determine the values of locale categories.)20090           **LC\_ALL**     If set to a non-empty string value, override the values of all the other  
20091                           internationalization variables.20092           **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
20093                           characters (for example, single-byte as opposed to multi-byte characters in  
20094                           arguments).20095           **LC\_MESSAGES**

20096                           Determine the locale that should be used to affect the format and contents of

20097 diagnostic messages written to standard error and informative messages written to  
 20098 standard output.

20099 xSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 20100 ASYNCHRONOUS EVENTS

20101 Default.

## 20102 STDOUT

20103 If the **-p** option is specified, the output shall consist of one line for each process ID:

20104 "%d\n", <process ID>

20105 Otherwise, if the **-l** option is not specified, the output shall be a series of lines of the form:

20106 "[%d] %c %s %s\n", <job-number>, <current>, <state>, <command>

20107 where the fields shall be as follows:

20108 <current> The character '+' identifies the job that would be used as a default for the *fg* or *bg*  
 20109 utilities; this job can also be specified using the *job\_id* %+ or "%%". The character  
 20110 '-' identifies the job that would become the default if the current default job were  
 20111 to exit; this job can also be specified using the *job\_id* %-. For other jobs, this field is  
 20112 a <space>. At most one job can be identified with '+' and at most one job can be  
 20113 identified with '-'. If there is any suspended job, then the current job shall be a  
 20114 suspended job. If there are at least two suspended jobs, then the previous job also  
 20115 shall be a suspended job.

20116 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*, and *kill*  
 20117 utilities. Using these utilities, the job can be identified by prefixing the job number  
 20118 with '% '.

20119 <state> One of the following strings (in the POSIX locale):

20120 **Running** Indicates that the job has not been suspended by a signal and has not  
 20121 exited.

20122 **Done** Indicates that the job completed and returned exit status zero.

20123 **Done(code)** Indicates that the job completed normally and that it exited with the  
 20124 specified non-zero exit status, *code*, expressed as a decimal number.

20125 **Stopped** Indicates that the job was suspended by the SIGTSTP signal.

20126 **Stopped (SIGTSTP)**

20127 Indicates that the job was suspended by the SIGTSTP signal.

20128 **Stopped (SIGSTOP)**

20129 Indicates that the job was suspended by the SIGSTOP signal.

20130 **Stopped (SIGTTIN)**

20131 Indicates that the job was suspended by the SIGTTIN signal.

20132 **Stopped (SIGTTOU)**

20133 Indicates that the job was suspended by the SIGTTOU signal.

20134 The implementation may substitute the string **Suspended** in place of **Stopped**. If  
 20135 the job was terminated by a signal, the format of <state> is unspecified, but it shall  
 20136 be visibly distinct from all of the other <state> formats shown here and shall  
 20137 indicate the name or description of the signal causing the termination.

20138           <*command*>   The associated command that was given to the shell.

20139           If the **-I** option is specified, a field containing the process group ID shall be inserted before the  
20140           <*state*> field. Also, more processes in a process group may be output on separate lines, using  
20141           only the process ID and <*command*> fields.

#### 20142 **STDERR**

20143           The standard error shall be used only for diagnostic messages.

#### 20144 **OUTPUT FILES**

20145           None.

#### 20146 **EXTENDED DESCRIPTION**

20147           None.

#### 20148 **EXIT STATUS**

20149           The following exit values shall be returned:

20150           0   Successful completion.

20151           >0   An error occurred.

#### 20152 **CONSEQUENCES OF ERRORS**

20153           Default.

#### 20154 **APPLICATION USAGE**

20155           The **-p** option is the only portable way to find out the process group of a job because different  
20156           implementations have different strategies for defining the process group of the job. Usage such  
20157           as *\$(jobs -p)* provides a way of referring to the process group of the job in an implementation-  
20158           independent way.

20159           The *jobs* utility does not work as expected when it is operating in its own utility execution  
20160           environment because that environment has no applicable jobs to manipulate. See the  
20161           APPLICATION USAGE section for *bg*. For this reason, *jobs* is generally implemented as a shell  
20162           regular built-in.

#### 20163 **EXAMPLES**

20164           None.

#### 20165 **RATIONALE**

20166           Both "%%" and "%+" are used to refer to the current job. Both forms are of equal validity—the  
20167           "%" mirroring "\$\$" and "%+" mirroring the output of *jobs*. Both forms reflect historical  
20168           practice of the KornShell and the C shell with job control.

20169           The job control features provided by *bg*, *fg*, and *jobs* are based on the KornShell. The standard  
20170           developers examined the characteristics of the C shell versions of these utilities and found that  
20171           differences exist. Despite widespread use of the C shell, the KornShell versions were selected for  
20172           this volume of IEEE Std 1003.1-2001 to maintain a degree of uniformity with the rest of the  
20173           KornShell features selected (such as the very popular command line editing features).

20174           The *jobs* utility is not dependent on the job control option, as are the seemingly related *bg* and *fg*  
20175           utilities because *jobs* is useful for examining background jobs, regardless of the condition of job  
20176           control. When the user has invoked a *set +m* command and job control has been turned off, *jobs*  
20177           can still be used to examine the background jobs associated with that current session. Similarly,  
20178           *kill* can then be used to kill background jobs with *kill% <background job number>*.

20179           The output for terminated jobs is left unspecified to accommodate various historical systems.  
20180           The following formats have been witnessed:

20181 1. **Killed**(*signal name*)

20182 2. *signal name*

20183 3. *signal name*(**coredump**)

20184 4. *signal description* – **core dumped**

20185 Most users should be able to understand these formats, although it means that applications have  
20186 trouble parsing them.

20187 The calculation of job IDs was not described since this would suggest an implementation, which  
20188 may impose unnecessary restrictions.

20189 In an early proposal, a **-n** option was included to “Display the status of jobs that have changed,  
20190 exited, or stopped since the last status report”. It was removed because the shell always writes  
20191 any changed status of jobs before each prompt.

20192 **FUTURE DIRECTIONS**

20193 None.

20194 **SEE ALSO**

20195 Section 2.12 (on page 61), *bg*, *fg*, *kill*, *wait*

20196 **CHANGE HISTORY**

20197 First released in Issue 4.

20198 **Issue 6**

20199 This utility is marked as part of the User Portability Utilities option.

20200 The JC shading is removed as job control is mandatory in this issue.



20201 **NAME**

20202           join — relational database operator

20203 **SYNOPSIS**

20204           join [-a *file\_number* | -v *file\_number*] [-e *string*] [-o *list*] [-t *char*]  
 20205                   [-1 *field*] [-2 *field*] *file1 file2*

20206 **DESCRIPTION**

20207           The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be  
 20208           written to the standard output.

20209           The join field is a field in each file on which the files are compared. The *join* utility shall write  
 20210           one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The  
 20211           output line by default shall consist of the join field, then the remaining fields from *file1*, then the  
 20212           remaining fields from *file2*. This format can be changed by using the -o option (see below). The  
 20213           -a option can be used to add unmatched lines to the output. The -v option can be used to output  
 20214           only unmatched lines.

20215           The files *file1* and *file2* shall be ordered in the collating sequence of sort -b on the fields on which  
 20216           they shall be joined, by default the first in each line. All selected output shall be written in the  
 20217           same collating sequence.

20218           The default input field separators shall be <blank>s. In this case, multiple separators shall count  
 20219           as one field separator, and leading separators shall be ignored. The default output field separator  
 20220           shall be a <space>.

20221           The field separator and collating sequence can be changed by using the -t option (see below).

20222           If the same key appears more than once in either file, all combinations of the set of remaining  
 20223           fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines  
 20224           encountered.

20225           If the input files are not in the appropriate collating sequence, the results are unspecified.

20226 **OPTIONS**

20227           The *join* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 20228           12.2, Utility Syntax Guidelines.

20229           The following options shall be supported:

20230           -a *file\_number*

20231                   Produce a line for each unpairable line in file *file\_number*, where *file\_number* is 1 or  
 20232                   2, in addition to the default output. If both -a1 and -a2 are specified, all unpairable  
 20233                   lines shall be output.

20234           -e *string*       Replace empty output fields in the list selected by -o with the string *string*.

20235           -o *list*         Construct the output line to comprise the fields specified in *list*, each element of  
 20236                   which shall have one of the following two forms:

20237                   1. *file\_number.field*, where *file\_number* is a file number and *field* is a decimal  
 20238                   integer field number

20239                   2. 0 (zero), representing the join field

20240           The elements of *list* shall be either comma-separated or <blank>-separated, as  
 20241           specified in Guideline 8 of the Base Definitions volume of IEEE Std 1003.1-2001,  
 20242           Section 12.2, Utility Syntax Guidelines. The fields specified by *list* shall be written  
 20243           for all selected output lines. Fields selected by *list* that do not appear in the input  
 20244           shall be treated as empty output fields. (See the -e option.) Only specifically

20245 requested fields shall be written. The application shall ensure that *list* is a single  
20246 command line argument.

20247 **-t char** Use character *char* as a separator, for both input and output. Every appearance of  
20248 *char* in a line shall be significant. When this option is specified, the collating  
20249 sequence shall be the same as *sort* without the **-b** option.

20250 **-v file\_number**  
20251 Instead of the default output, produce a line only for each unpairable line in  
20252 *file\_number*, where *file\_number* is 1 or 2. If both **-v1** and **-v2** are specified, all  
20253 unpairable lines shall be output.

20254 **-1 field** Join on the *field*th field of file 1. Fields are decimal integers starting with 1.

20255 **-2 field** Join on the *field*th field of file 2. Fields are decimal integers starting with 1.

## 20256 OPERANDS

20257 The following operands shall be supported:

20258 *file1, file2* A pathname of a file to be joined. If either of the *file1* or *file2* operands is '-', the  
20259 standard input shall be used in its place.

## 20260 STDIN

20261 The standard input shall be used only if the *file1* or *file2* operand is '-'. See the INPUT FILES  
20262 section.

## 20263 INPUT FILES

20264 The input files shall be text files.

## 20265 ENVIRONMENT VARIABLES

20266 The following environment variables shall affect the execution of *join*:

20267 **LANG** Provide a default value for the internationalization variables that are unset or null.  
20268 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
20269 Internationalization Variables for the precedence of internationalization variables  
20270 used to determine the values of locale categories.)

20271 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
20272 internationalization variables.

20273 **LC\_COLLATE**  
20274 Determine the locale of the collating sequence *join* expects to have been used when  
20275 the input files were sorted.

20276 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
20277 characters (for example, single-byte as opposed to multi-byte characters in  
20278 arguments and input files).

20279 **LC\_MESSAGES**  
20280 Determine the locale that should be used to affect the format and contents of  
20281 diagnostic messages written to standard error.

20282 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

## 20283 ASYNCHRONOUS EVENTS

20284 Default.

20285 **STDOUT**

20286 The *join* utility output shall be a concatenation of selected character fields. When the **-o** option  
 20287 is not specified, the output shall be:

20288 "%s%s%s\n", <join field>, <other file1 fields>,  
 20289 <other file2 fields>

20290 If the join field is not the first field in a file, the <other file fields> for that file shall be:

20291 <fields preceding join field>, <fields following join field>

20292 When the **-o** option is specified, the output format shall be:

20293 "%s\n", <concatenation of fields>

20294 where the concatenation of fields is described by the **-o** option, above.

20295 For either format, each field (except the last) shall be written with its trailing separator character.  
 20296 If the separator is the default (<blank>s), a single <space> shall be written after each field  
 20297 (except the last).

20298 **STDERR**

20299 The standard error shall be used only for diagnostic messages.

20300 **OUTPUT FILES**

20301 None.

20302 **EXTENDED DESCRIPTION**

20303 None.

20304 **EXIT STATUS**

20305 The following exit values shall be returned:

20306 0 All input files were output successfully.

20307 >0 An error occurred.

20308 **CONSEQUENCES OF ERRORS**

20309 Default.

20310 **APPLICATION USAGE**

20311 Pathnames consisting of numeric digits or of the form *string.string* should not be specified  
 20312 directly following the **-o** list.

20313 **EXAMPLES**

20314 The **-o 0** field essentially selects the union of the join fields. For example, given file **phone**:

|       |         |                 |
|-------|---------|-----------------|
| 20315 | !Name   | Phone Number    |
| 20316 | Don     | +1 123-456-7890 |
| 20317 | Hal     | +1 234-567-8901 |
| 20318 | Yasushi | +2 345-678-9012 |

20319 and file **fax**:

|       |         |                 |
|-------|---------|-----------------|
| 20320 | !Name   | Fax Number      |
| 20321 | Don     | +1 123-456-7899 |
| 20322 | Keith   | +1 456-789-0122 |
| 20323 | Yasushi | +2 345-678-9011 |

20324 (where the large expanses of white space are meant to each represent a single <tab>), the  
 20325 command:

20326 `join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax`  
 20327 would produce:

| 20328 | !Name   | Phone Number    | Fax Number      |
|-------|---------|-----------------|-----------------|
| 20329 | Don     | +1 123-456-7890 | +1 123-456-7899 |
| 20330 | Hal     | +1 234-567-8901 | (unknown)       |
| 20331 | Keith   | (unknown)       | +1 456-789-0122 |
| 20332 | Yasushi | +2 345-678-9012 | +2 345-678-9011 |

20333 Multiple instances of the same key will produce combinatorial results. The following:

20334 fa:  
 20335     a x  
 20336     a y  
 20337     a z  
 20338 fb:  
 20339     a p

20340 will produce:

20341 a x p  
 20342 a y p  
 20343 a z p

20344 And the following:

20345 fa:  
 20346     a b c  
 20347     a d e  
 20348 fb:  
 20349     a w x  
 20350     a y z  
 20351     a o p

20352 will produce:

20353 a b c w x  
 20354 a b c y z  
 20355 a b c o p  
 20356 a d e w x  
 20357 a d e y z  
 20358 a d e o p

#### 20359 RATIONALE

20360 The `-e` option is only effective when used with `-o` because, unless specific fields are identified  
 20361 using `-o`, *join* is not aware of what fields might be empty. The exception to this is the join field,  
 20362 but identifying an empty join field with the `-e` string is not historical practice and some scripts  
 20363 might break if this were changed.

20364 The 0 field in the `-o` list was adopted from the Tenth Edition version of *join* to satisfy  
 20365 international objections that the *join* in the base documents does not support the “full join” or  
 20366 “outer join” described in relational database literature. Although it has been possible to include  
 20367 a join field in the output (by default, or by field number using `-o`), the join field could not be  
 20368 included for an unpaired line selected by `-a`. The `-o 0` field essentially selects the union of the  
 20369 join fields.

20370 This sort of outer join was not possible with the *join* commands in the base documents. The `-o 0`  
 20371 field was chosen because it is an upwards-compatible change for applications. An alternative

20372 was considered: have the join field represent the union of the fields in the files (where they are  
20373 identical for matched lines, and one or both are null for unmatched lines). This was not adopted  
20374 because it would break some historical applications.

20375 The ability to specify *file2* as – is not historical practice; it was added for completeness.

20376 The –v option is not historical practice, but was considered necessary because it permitted the  
20377 writing of *only* those lines that do not match on the join field, as opposed to the –a option, which  
20378 prints both lines that do and do not match. This additional facility is parallel with the –v option  
20379 of *grep*.

20380 Some historical implementations have been encountered where a blank line in one of the input  
20381 files was considered to be the end of the file; the description in this volume of  
20382 IEEE Std 1003.1-2001 does not cite this as an allowable case.

#### 20383 **FUTURE DIRECTIONS**

20384 None.

#### 20385 **SEE ALSO**

20386 *awk, comm, sort, uniq*

#### 20387 **CHANGE HISTORY**

20388 First released in Issue 2.

#### 20389 **Issue 6**

20390 The obsolescent –j options and the multi-argument –o option are withdrawn in this issue.

20391 The normative text is reworded to avoid use of the term “must” for application requirements.

20392 **NAME**

20393       kill — terminate or signal processes

20394 **SYNOPSIS**20395       kill -s *signal\_name* *pid* ...20396       kill -l [*exit\_status*]20397 xSI     kill [-*signal\_name*] *pid* ...20398       kill [-*signal\_number*] *pid* ...

20399

20400 **DESCRIPTION**20401       The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.20402       For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill()* function  
20403       defined in the System Interfaces volume of IEEE Std 1003.1-2001 called with the following  
20404       arguments:

- 20405       • The value of the *pid* operand shall be used as the *pid* argument.
- 20406       • The *sig* argument is the value specified by the -s option, -*signal\_number* option, or the
- 20407        -*signal\_name* option, or by SIGTERM, if none of these options is specified.

20408 **OPTIONS**

20409       The *kill* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
20410 xSI     12.2, Utility Syntax Guidelines, except that in the last two SYNOPSIS forms, the -*signal\_number*  
20411       and -*signal\_name* options are usually more than a single character.

20412       The following options shall be supported:

20413       -l       (The letter ell.) Write all values of *signal\_name* supported by the implementation, if  
20414       no operand is given. If an *exit\_status* operand is given and it is a value of the ' ? '   
20415       shell special parameter (see Section 2.5.2 (on page 34) and *wait*) corresponding to a  
20416       process that was terminated by a signal, the *signal\_name* corresponding to the  
20417       signal that terminated the process shall be written. If an *exit\_status* operand is  
20418       given and it is the unsigned decimal integer value of a signal number, the  
20419       *signal\_name* (the symbolic constant name without the **SIG** prefix defined in the  
20420       Base Definitions volume of IEEE Std 1003.1-2001) corresponding to that signal  
20421       shall be written. Otherwise, the results are unspecified.

20422       -s *signal\_name*

20423       Specify the signal to send, using one of the symbolic names defined in the  
20424       <**signal.h**> header. Values of *signal\_name* shall be recognized in a case-independent  
20425       fashion, without the **SIG** prefix. In addition, the symbolic name 0 shall be  
20426       recognized, representing the signal value zero. The corresponding signal shall be  
20427       sent instead of SIGTERM.

20428 xSI     -*signal\_name*20429       Equivalent to -s *signal\_name*.20430 xSI     -*signal\_number*

20431       Specify a non-negative decimal integer, *signal\_number*, representing the signal to  
20432       be used instead of SIGTERM, as the *sig* argument in the effective call to *kill()*. The  
20433       correspondence between integer values and the *sig* value used is shown in the  
20434       following list.

20435       The effects of specifying any *signal\_number* other than those listed below are  
20436       undefined.

|       |                                                                                                 |         |  |
|-------|-------------------------------------------------------------------------------------------------|---------|--|
| 20437 | 0                                                                                               | 0       |  |
| 20438 | 1                                                                                               | SIGHUP  |  |
| 20439 | 2                                                                                               | SIGINT  |  |
| 20440 | 3                                                                                               | SIGQUIT |  |
| 20441 | 6                                                                                               | SIGABRT |  |
| 20442 | 9                                                                                               | SIGKILL |  |
| 20443 | 14                                                                                              | SIGALRM |  |
| 20444 | 15                                                                                              | SIGTERM |  |
| 20445 | If the first argument is a negative integer, it shall be interpreted as a <i>-signal_number</i> |         |  |
| 20446 | option, not as a negative <i>pid</i> operand specifying a process group.                        |         |  |

#### 20447 OPERANDS

20448 The following operands shall be supported:

|       |                    |                                                                                              |
|-------|--------------------|----------------------------------------------------------------------------------------------|
| 20449 | <i>pid</i>         | One of the following:                                                                        |
| 20450 |                    | 1. A decimal integer specifying a process or process group to be signaled. The               |
| 20451 |                    | process or processes selected by positive, negative, and zero values of the <i>pid</i>       |
| 20452 |                    | operand shall be as described for the <i>kill()</i> function. If process number 0 is         |
| 20453 |                    | specified, all processes in the current process group shall be signaled. For the             |
| 20454 |                    | effects of negative <i>pid</i> numbers, see the <i>kill()</i> function defined in the System |
| 20455 |                    | Interfaces volume of IEEE Std 1003.1-2001. If the first <i>pid</i> operand is negative,      |
| 20456 |                    | it should be preceded by "--" to keep it from being interpreted as an option.                |
| 20457 |                    | 2. A job control job ID (see the Base Definitions volume of IEEE Std 1003.1-2001,            |
| 20458 |                    | Section 3.203, Job Control Job ID) that identifies a background process group                |
| 20459 |                    | to be signaled. The job control job ID notation is applicable only for                       |
| 20460 |                    | invocations of <i>kill</i> in the current shell execution environment; see Section 2.12      |
| 20461 |                    | (on page 61).                                                                                |
| 20462 | <i>exit_status</i> | A decimal integer specifying a signal number or the exit status of a process                 |
| 20463 |                    | terminated by a signal.                                                                      |

#### 20464 STDIN

20465 Not used.

#### 20466 INPUT FILES

20467 None.

#### 20468 ENVIRONMENT VARIABLES

20469 The following environment variables shall affect the execution of *kill*:

|       |                 |                                                                                        |
|-------|-----------------|----------------------------------------------------------------------------------------|
| 20470 | <i>LANG</i>     | Provide a default value for the internationalization variables that are unset or null. |
| 20471 |                 | (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,                 |
| 20472 |                 | Internationalization Variables for the precedence of internationalization variables    |
| 20473 |                 | used to determine the values of locale categories.)                                    |
| 20474 | <i>LC_ALL</i>   | If set to a non-empty string value, override the values of all the other               |
| 20475 |                 | internationalization variables.                                                        |
| 20476 | <i>LC_CTYPE</i> | Determine the locale for the interpretation of sequences of bytes of text data as      |
| 20477 |                 | characters (for example, single-byte as opposed to multi-byte characters in            |
| 20478 |                 | arguments).                                                                            |

20479 **LC\_MESSAGES**  
 20480 Determine the locale that should be used to affect the format and contents of  
 20481 diagnostic messages written to standard error.

20482 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

20483 **ASYNCHRONOUS EVENTS**  
 20484 Default.

20485 **STDOUT**  
 20486 When the **-l** option is not specified, the standard output shall not be used.

20487 When the **-l** option is specified, the symbolic name of each signal shall be written in the  
 20488 following format:  
 20489 "%s%c", <signal\_name>, <separator>  
 20490 where the <signal\_name> is in uppercase, without the **SIG** prefix, and the <separator> shall be  
 20491 either a <newline> or a <space>. For the last signal written, <separator> shall be a <newline>.

20492 When both the **-l** option and *exit\_status* operand are specified, the symbolic name of the  
 20493 corresponding signal shall be written in the following format:  
 20494 "%s\n", <signal\_name>

20495 **STDERR**  
 20496 The standard error shall be used only for diagnostic messages.

20497 **OUTPUT FILES**  
 20498 None.

20499 **EXTENDED DESCRIPTION**  
 20500 None.

20501 **EXIT STATUS**  
 20502 The following exit values shall be returned:  
 20503 0 At least one matching process was found for each *pid* operand, and the specified signal was  
 20504 successfully processed for at least one matching process.  
 20505 >0 An error occurred.

20506 **CONSEQUENCES OF ERRORS**  
 20507 Default.

20508 **APPLICATION USAGE**  
 20509 Process numbers can be found by using *ps*.

20510 The job control job ID notation is not required to work as expected when *kill* is operating in its  
 20511 own utility execution environment. In either of the following examples:  
 20512 nohup kill %1 &  
 20513 system("kill %1");  
 20514 the *kill* operates in a different environment and does not share the shell's understanding of job  
 20515 numbers.

20516 **EXAMPLES**  
 20517 Any of the commands:  
 20518 kill -9 100 -165  
 20519 kill -s kill 100 -165  
 20520 kill -s KILL 100 -165



20521 sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose  
 20522 process group ID is 165, assuming the sending process has permission to send that signal to the  
 20523 specified processes, and that they exist.

20524 The System Interfaces volume of IEEE Std 1003.1-2001 and this volume of IEEE Std 1003.1-2001  
 20525 do not require specific signal numbers for any *signal\_names*. Even the *-signal\_number* option  
 20526 provides symbolic (although numeric) names for signals. If a process is terminated by a signal,  
 20527 its exit status indicates the signal that killed it, but the exact values are not specified. The *kill -l*  
 20528 option, however, can be used to map decimal signal numbers and exit status values into the  
 20529 name of a signal. The following example reports the status of a terminated job:

```
20530 job
20531 stat=$?
20532 if [$stat -eq 0]
20533 then
20534 echo job completed successfully.
20535 elif [$stat -gt 128]
20536 then
20537 echo job terminated by signal SIG$(kill -l $stat).
20538 else
20539 echo job terminated with error code $stat.
20540 fi
```

20541 To send the default signal to a process group (say 123), an application should use a command  
 20542 similar to one of the following:

```
20543 kill -TERM -123
20544 kill -- -123
```

#### 20545 RATIONALE

20546 The *-l* option originated from the C shell, and is also implemented in the KornShell. The C shell  
 20547 output can consist of multiple output lines because the signal names do not always fit on a  
 20548 single line on some terminal screens. The KornShell output also included the implementation-  
 20549 defined signal numbers and was considered by the standard developers to be too difficult for  
 20550 scripts to parse conveniently. The specified output format is intended not only to accommodate  
 20551 the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing  
 20552 on systems for which this is appropriate.

20553 An early proposal invented the name SIGNULL as a *signal\_name* for signal 0 (used by the System  
 20554 Interfaces volume of IEEE Std 1003.1-2001 to test for the existence of a process without sending it  
 20555 a signal). Since the *signal\_name* 0 can be used in this case unambiguously, SIGNULL has been  
 20556 removed.

20557 An early proposal also required symbolic *signal\_names* to be recognized with or without the **SIG**  
 20558 prefix. Historical versions of *kill* have not written the **SIG** prefix for the *-l* option and have not  
 20559 recognized the **SIG** prefix on *signal\_names*. Since neither applications portability nor ease-of-use  
 20560 would be improved by requiring this extension, it is no longer required.

20561 To avoid an ambiguity of an initial negative number argument specifying either a signal number  
 20562 or a process group, IEEE Std 1003.1-2001 mandates that it is always considered the former by  
 20563 implementations that support the XSI option. It also requires that conforming applications  
 20564 always use the "--" options terminator argument when specifying a process group, unless an  
 20565 option is also specified.

20566 The *-s* option was added in response to international interest in providing some form of *kill* that  
 20567 meets the Utility Syntax Guidelines.

20568           The job control job ID notation is not required to work as expected when *kill* is operating in its  
20569           own utility execution environment. In either of the following examples:

```
20570 nohup kill %1 &
20571 system("kill %1");
```

20572           the *kill* operates in a different environment and does not understand how the shell has managed  
20573           its job numbers.

#### 20574 **FUTURE DIRECTIONS**

20575           None.

#### 20576 **SEE ALSO**

20577           Chapter 2 (on page 29), *ps*, *wait*, the System Interfaces volume of IEEE Std 1003.1-2001, *kill*( ), the  
20578           Base Definitions volume of IEEE Std 1003.1-2001, <**signal.h**>

#### 20579 **CHANGE HISTORY**

20580           First released in Issue 2.

#### 20581 **Issue 6**

20582           The obsolescent versions of the SYNOPSIS are turned into non-obsolescent features of the XSI  
20583           option, corresponding to a similar change in the *trap* special built-in.

20584 **NAME**20585       lex — generate programs for lexical tasks (**DEVELOPMENT**)20586 **SYNOPSIS**20587 CD       lex [-t] [-n|-v] [*file* ...]

20588

20589 **DESCRIPTION**

20590       The *lex* utility shall generate C programs to be used in lexical processing of character input, and  
 20591       that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code  
 20592       and conform to the ISO C standard. Usually, the *lex* utility shall write the program it generates to  
 20593       the file **lex.yy.c**; the state of this file is unspecified if *lex* exits with a non-zero exit status. See the  
 20594       EXTENDED DESCRIPTION section for a complete description of the *lex* input language.

20595 **OPTIONS**

20596       The *lex* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 20597       Utility Syntax Guidelines.

20598       The following options shall be supported:

20599       **-n**           Suppress the summary of statistics usually written with the **-v** option. If no table  
 20600       sizes are specified in the *lex* source code and the **-v** option is not specified, then **-n**  
 20601       is implied.

20602       **-t**           Write the resulting program to standard output instead of **lex.yy.c**.

20603       **-v**           Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*  
 20604       table sizes in **Definitions in lex** (on page 537).) If the **-t** option is specified and **-n**  
 20605       is not specified, this report shall be written to standard error. If table sizes are  
 20606       specified in the *lex* source code, and if the **-n** option is not specified, the **-v** option  
 20607       may be enabled.

20608 **OPERANDS**

20609       The following operand shall be supported:

20610       *file*        A pathname of an input file. If more than one such *file* is specified, all files shall be  
 20611       concatenated to produce a single *lex* program. If no *file* operands are specified, or if  
 20612       a *file* operand is '-', the standard input shall be used.

20613 **STDIN**

20614       The standard input shall be used if no *file* operands are specified, or if a *file* operand is '-'. See  
 20615       INPUT FILES.

20616 **INPUT FILES**

20617       The input files shall be text files containing *lex* source code, as described in the EXTENDED  
 20618       DESCRIPTION section.

20619 **ENVIRONMENT VARIABLES**

20620       The following environment variables shall affect the execution of *lex*:

20621       **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 20622       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 20623       Internationalization Variables for the precedence of internationalization variables  
 20624       used to determine the values of locale categories.)

20625       **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 20626       internationalization variables.

20627       **LC\_COLLATE**

20628       Determine the locale for the behavior of ranges, equivalence classes, and multi-

20629 character collating elements within regular expressions. If this variable is not set to  
20630 the POSIX locale, the results are unspecified.

20631 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
20632 characters (for example, single-byte as opposed to multi-byte characters in  
20633 arguments and input files), and the behavior of character classes within regular  
20634 expressions. If this variable is not set to the POSIX locale, the results are  
20635 unspecified.

20636 **LC\_MESSAGES**  
20637 Determine the locale that should be used to affect the format and contents of  
20638 diagnostic messages written to standard error.

20639 **NSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

## 20640 ASYNCHRONOUS EVENTS

20641 Default.

## 20642 STDOUT

20643 If the **-t** option is specified, the text file of C source code output of *lex* shall be written to  
20644 standard output.

20645 If the **-t** option is not specified:

- 20646 • Implementation-defined informational, error, and warning messages concerning the contents  
20647 of *lex* source code input shall be written to either the standard output or standard error.
- 20648 • If the **-v** option is specified and the **-n** option is not specified, *lex* statistics shall also be  
20649 written to either the standard output or standard error, in an implementation-defined format.  
20650 These statistics may also be generated if table sizes are specified with a '%' operator in the  
20651 *Definitions* section, as long as the **-n** option is not specified.

## 20652 STDERR

20653 If the **-t** option is specified, implementation-defined informational, error, and warning messages  
20654 concerning the contents of *lex* source code input shall be written to the standard error.

20655 If the **-t** option is not specified:

- 20656 1. Implementation-defined informational, error, and warning messages concerning the  
20657 contents of *lex* source code input shall be written to either the standard output or standard  
20658 error.
- 20659 2. If the **-v** option is specified and the **-n** option is not specified, *lex* statistics shall also be  
20660 written to either the standard output or standard error, in an implementation-defined  
20661 format. These statistics may also be generated if table sizes are specified with a '%'   
20662 operator in the *Definitions* section, as long as the **-n** option is not specified.

## 20663 OUTPUT FILES

20664 A text file containing C source code shall be written to **lex.yy.c**, or to the standard output if the  
20665 **-t** option is present.

## 20666 EXTENDED DESCRIPTION

20667 Each input file shall contain *lex* source code, which is a table of regular expressions with  
20668 corresponding actions in the form of C program fragments.

20669 When **lex.yy.c** is compiled and linked with the *lex* library (using the **-ll** operand with *c99*), the  
20670 resulting program shall read character input from the standard input and shall partition it into  
20671 strings that match the given expressions.

20672 When an expression is matched, these actions shall occur:

- 20673 • The input string that was matched shall be left in *ytext* as a null-terminated string; *ytext*  
20674 shall either be an external character array or a pointer to a character string. As explained in  
20675 **Definitions in lex**, the type can be explicitly selected using the **%array** or **%pointer**  
20676 declarations, but the default is implementation-defined.
- 20677 • The external **int** *yyleng* shall be set to the length of the matching string.
- 20678 • The expression's corresponding program fragment, or action, shall be executed.

20679 During pattern matching, *lex* shall search the set of patterns for the single longest possible  
20680 match. Among rules that match the same number of characters, the rule given first shall be  
20681 chosen.

20682 The general format of *lex* source shall be:

```
20683 Definitions
20684 %%
20685 Rules
20686 %%
20687 UserSubroutines
```

20688 The first "%%" is required to mark the beginning of the rules (regular expressions and actions);  
20689 the second "%%" is required only if user subroutines follow.

20690 Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program  
20691 fragment and shall be copied to the external definition area of the **lex.yy.c** file. Similarly,  
20692 anything in the *Definitions* section included between delimiter lines containing only "%{" and  
20693 "%}" shall also be copied unchanged to the external definition area of the **lex.yy.c** file.

20694 Any such input (beginning with a <blank> or within "%{" and "%}" delimiter lines) appearing  
20695 at the beginning of the *Rules* section before any rules are specified shall be written to **lex.yy.c**  
20696 after the declarations of variables for the *yylex()* function and before the first line of code in  
20697 *yylex()*. Thus, user variables local to *yylex()* can be declared here, as well as application code to  
20698 execute upon entry to *yylex()*.

20699 The action taken by *lex* when encountering any input beginning with a <blank> or within "%{"  
20700 and "%}" delimiter lines appearing in the *Rules* section but coming after one or more rules is  
20701 undefined. The presence of such input may result in an erroneous definition of the *yylex()*  
20702 function.

## 20703 **Definitions in lex**

20704 *Definitions* appear before the first "%%" delimiter. Any line in this section not contained between  
20705 "%{" and "%}" lines and not beginning with a <blank> shall be assumed to define a *lex*  
20706 substitution string. The format of these lines shall be:

```
20707 name substitute
```

20708 If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is  
20709 undefined. The string *substitute* shall replace the string {*name*} when it is used in a rule. The *name*  
20710 string shall be recognized in this context only when the braces are provided and when it does  
20711 not appear within a bracket expression or within double-quotes.

20712 In the *Definitions* section, any line beginning with a '%' (percent sign) character and followed by  
20713 an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions.  
20714 Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall define  
20715 a set of exclusive start conditions. When the generated scanner is in a %s state, patterns with no

state specified shall be also active; in a %x state, such patterns shall not be active. The rest of the line, after the first word, shall be considered to be one or more <blank>-separated names of start conditions. Start condition names shall be constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in **Regular Expressions in lex** (on page 539).

Implementations shall accept either of the following two mutually-exclusive declarations in the *Definitions* section:

**%array**      Declare the type of *yytext* to be a null-terminated character array.

**%pointer**    Declare the type of *yytext* to be a pointer to a null-terminated character string.

The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of the scanner source file (that is, via an **extern**), the application shall include the appropriate **%array** or **%pointer** declaration in the scanner source file.

Implementations shall accept declarations in the *Definitions* section for setting certain internal table sizes. The declarations are shown in the following table.

**Table 4-9** Table Size Declarations in *lex*

| Declaration        | Description                        | Minimum Value |
|--------------------|------------------------------------|---------------|
| <b>%p</b> <i>n</i> | Number of positions                | 2 500         |
| <b>%n</b> <i>n</i> | Number of states                   | 500           |
| <b>%a</b> <i>n</i> | Number of transitions              | 2 000         |
| <b>%e</b> <i>n</i> | Number of parse tree nodes         | 1 000         |
| <b>%k</b> <i>n</i> | Number of packed character classes | 1 000         |
| <b>%o</b> <i>n</i> | Size of the output array           | 3 000         |

In the table, *n* represents a positive decimal integer, preceded by one or more <blank>s. The exact meaning of these table size numbers is implementation-defined. The implementation shall document how these numbers affect the *lex* utility and how they are related to any output that may be generated by the implementation should limitations be encountered during the execution of *lex*. It shall be possible to determine from this output which of the table size values needs to be modified to permit *lex* to successfully generate tables for the input language. The values in the column Minimum Value represent the lowest values conforming implementations shall provide.

### Rules in *lex*

The rules in *lex* source files are a table in which the left column contains regular expressions and the right column contains actions (C program fragments) to be executed when the expressions are recognized.

*ERE action*

*ERE action*

...

The extended regular expression (ERE) portion of a row shall be separated from *action* by one or more <blank>s. A regular expression containing <blank>s shall be recognized under one of the following conditions:

- The entire expression appears within double-quotes.
- The <blank>s appear within double-quotes or square brackets.

- Each <blank> is preceded by a backslash character.

## 20759 User Subroutines in lex

20760 Anything in the user subroutines section shall be copied to **lex.yy.c** following `yylex()`.

## 20761 Regular Expressions in lex

20762 The *lex* utility shall support the set of extended regular expressions (see the Base Definitions  
20763 volume of IEEE Std 1003.1-2001, Section 9.4, Extended Regular Expressions), with the following  
20764 additions and exceptions to the syntax:

20765 " . . . " Any string enclosed in double-quotes shall represent the characters within the  
20766 double-quotes as themselves, except that backslash escapes (which appear in the  
20767 following table) shall be recognized. Any backslash-escape sequence shall be  
20768 terminated by the closing quote. For example, "\01""1" represents a single  
20769 string: the octal value 1 followed by the character '1'.

20770 <state>*r*, <state1, state2, . . .>*r*

20771 The regular expression *r* shall be matched only when the program is in one of the  
20772 start conditions indicated by *state*, *state1*, and so on; see **Actions in lex** (on page  
20773 541). (As an exception to the typographical conventions of the rest of this volume  
20774 of IEEE Std 1003.1-2001, in this case <state> does not represent a metavariable, but  
20775 the literal angle-bracket characters surrounding a symbol.) The start condition  
20776 shall be recognized as such only at the beginning of a regular expression.

20777 *r*/*x* The regular expression *r* shall be matched only if it is followed by an occurrence of  
20778 regular expression *x* (*x* is the instance of trailing context, further defined below).  
20779 The token returned in *yytext* shall only match *r*. If the trailing portion of *r* matches  
20780 the beginning of *x*, the result is unspecified. The *r* expression cannot include  
20781 further trailing context or the '\$' (match-end-of-line) operator; *x* cannot include  
20782 the '^' (match-beginning-of-line) operator, nor trailing context, nor the '\$'  
20783 operator. That is, only one occurrence of trailing context is allowed in a *lex* regular  
20784 expression, and the '^' operator only can be used at the beginning of such an  
20785 expression.

20786 {*name*} When *name* is one of the substitution symbols from the *Definitions* section, the  
20787 string, including the enclosing braces, shall be replaced by the *substitute* value. The  
20788 *substitute* value shall be treated in the extended regular expression as if it were  
20789 enclosed in parentheses. No substitution shall occur if {*name*} occurs within a  
20790 bracket expression or within double-quotes.

20791 Within an ERE, a backslash character shall be considered to begin an escape sequence as  
20792 specified in the table in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File  
20793 Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape  
20794 sequences in the following table shall be recognized.

20795 A literal <newline> cannot occur within an ERE; the escape sequence '\n' can be used to  
20796 represent a <newline>. A <newline> shall not be matched by a period operator.

20797

**Table 4-10** Escape Sequences in *lex*

20798

20799

20800

20801

20802

20803

20804

20805

20806

20807

20808

20809

20810

20811

20812

20813

20814

20815

20816

20817

20818

20819

20820

20821

20822

20823

20824

20825

20826

20827

| Escape Sequence       | Description                                                                                                                                                                                                                                                                                                                                           | Meaning                                                                                                                                                                                                                                                                                                                                                                           |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\digits</code>  | A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.                                                                                                                                 | The character whose encoding is represented by the one, two, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <code>'\'</code> for each byte. |
| <code>\xdigits</code> | A backslash character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.                                                                                                                                  | The character whose encoding is represented by the hexadecimal integer.                                                                                                                                                                                                                                                                                                           |
| <code>\c</code>       | A backslash character followed by any character not described in this table or in the table in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation ( <code>'\'</code> , <code>'\a'</code> , <code>'\b'</code> , <code>'\f'</code> , <code>'\n'</code> , <code>'\r'</code> , <code>'\t'</code> , <code>'\v'</code> ). | The character <code>'c'</code> , unchanged.                                                                                                                                                                                                                                                                                                                                       |

20828

20829

20830

**Note:** If a `'\x'` sequence needs to be immediately followed by a hexadecimal digit character, a sequence such as `"\x1"1"` can be used, which represents a character containing the value 1, followed by the character `'1'`.

20831

20832

20833

20834

The order of precedence given to extended regular expressions for *lex* differs from that specified in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.4, Extended Regular Expressions. The order of precedence for *lex* shall be as shown in the following table, from high to low.

20835

20836

20837

20838

20839

**Note:** The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context, and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.



Table 4-11 ERE Precedence in *lex*

| Extended Regular Expression       | Precedence            |
|-----------------------------------|-----------------------|
| collation-related bracket symbols | [ = ] [ : : ] [ . . ] |
| escaped characters                | \<special character>  |
| bracket expression                | [ ]                   |
| quoting                           | " . . . "             |
| grouping                          | ( )                   |
| definition                        | { name }              |
| single-character RE duplication   | * + ?                 |
| concatenation                     |                       |
| interval expression               | { m , n }             |
| alternation                       |                       |

The ERE anchoring operators '**^**' and '**\$**' do not appear in the table. With *lex* regular expressions, these operators are restricted in their use: the '**^**' operator can only be used at the beginning of an entire regular expression, and the '**\$**' operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern "**(^abc) | (def\$)**" is undefined; it can instead be written as two separate rules, one with the regular expression "**^abc**" and one with "**def\$**", which share a common action via the special '**|**' action (see below). If the pattern were written "**^abc|def\$**", it would match either "abc" or "def" on a line by itself.

Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex* implementations. An example of embedded anchoring would be for patterns such as "**(^| )foo( |\$)**" to match "foo" when it exists as a complete word. This functionality can be obtained using existing *lex* features:

```
^foo/[\n] |
" foo"/[\n] /* Found foo as a separate word. */
```

Note also that '**\$**' is a form of trailing context (it is equivalent to "**/\n**") and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

The additional regular expressions trailing-context operator '**/'**' can be used as an ordinary character if presented within double-quotes, "**/**"; preceded by a backslash, "**\/**"; or within a bracket expression, "**[/]**". The start-condition '**<**' and '**>**' operators shall be special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they shall be treated as ordinary characters.

### Actions in *lex*

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions. The empty C statement '**;**' shall be a valid action; any string in the **lex.yy.c** input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action shall not be valid, and the action *lex* takes in such a condition is undefined.

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

```
ERE <one or more blanks> { program statement
 program statement }
```

20885 The default action when a string in the input to a **lex.yy.c** program is not matched by any  
 20886 expression shall be to copy the string to the output. Because the default behavior of a program  
 20887 generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that  
 20888 has just "%%" shall generate a C program that simply copies the input to the output unchanged.

20889 Four special actions shall be available:

20890 | ECHO; REJECT; BEGIN

20891 | The action ' | ' means that the action for the next rule is the action for this rule.  
 20892 Unlike the other three actions, ' | ' cannot be enclosed in braces or be semicolon-  
 20893 terminated; the application shall ensure that it is specified alone, with no other  
 20894 actions.

20895 **ECHO;** Write the contents of the string *yytext* on the output.

20896 **REJECT;** Usually only a single expression is matched by a given string in the input. **REJECT**  
 20897 means "continue to the next expression that matches the current input", and shall  
 20898 cause whatever rule was the second choice after the current rule to be executed for  
 20899 the same input. Thus, multiple rules can be matched and executed for one input  
 20900 string or overlapping input strings. For example, given the regular expressions  
 20901 "xyz" and "xy" and the input "xyz", usually only the regular expression "xyz"  
 20902 would match. The next attempted match would start after **z**. If the last action in the  
 20903 "xyz" rule is **REJECT**, both this rule and the "xy" rule would be executed. The  
 20904 **REJECT** action may be implemented in such a fashion that flow of control does not  
 20905 continue after it, as if it were equivalent to a **goto** to another part of *yylex()*. The  
 20906 use of **REJECT** may result in somewhat larger and slower scanners.

20907 **BEGIN** The action:

20908 BEGIN *newstate*;

20909 switches the state (start condition) to *newstate*. If the string *newstate* has not been  
 20910 declared previously as a start condition in the *Definitions* section, the results are  
 20911 unspecified. The initial state is indicated by the digit '0' or the token **INITIAL**.

20912 The functions or macros described below are accessible to user code included in the *lex* input. It  
 20913 is unspecified whether they appear in the C code output of *lex*, or are accessible only through the  
 20914 **-ll** operand to *c99* (the *lex* library).

20915 **int yylex(void)**  
 20916 Performs lexical analysis on the input; this is the primary function generated by the *lex*  
 20917 utility. The function shall return zero when the end of input is reached; otherwise, it shall  
 20918 return non-zero values (tokens) determined by the actions that are selected.

20919 **int yymore(void)**  
 20920 When called, indicates that when the next input string is recognized, it is to be appended to  
 20921 the current value of *yytext* rather than replacing it; the value in *yyleng* shall be adjusted  
 20922 accordingly.

20923 **int yyless(int n)**  
 20924 Retains *n* initial characters in *yytext*, NUL-terminated, and treats the remaining characters  
 20925 as if they had not been read; the value in *yyleng* shall be adjusted accordingly.

20926 **int input(void)**  
 20927 Returns the next character from the input, or zero on end-of-file. It shall obtain input from  
 20928 the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning  
 20929 has begun, the effect of altering the value of *yyin* is undefined. The character read shall be  
 20930 removed from the input stream of the scanner without any processing by the scanner.

20931 **int unput(int c)**  
 20932 Returns the character 'c' to the input; *yytext* and *yylen* are undefined until the next  
 20933 expression is matched. The result of using *unput()* for more characters than have been input  
 20934 is unspecified.

20935 The following functions shall appear only in the *lex* library accessible through the **-ll** operand;  
 20936 they can therefore be redefined by a conforming application:

20937 **int yywrap(void)**  
 20938 Called by *yylex()* at end-of-file; the default *yywrap()* shall always return 1. If the application  
 20939 requires *yylex()* to continue processing with another source of input, then the application  
 20940 can include a function *yywrap()*, which associates another file with the external variable  
 20941 **FILE \*yyin** and shall return a value of zero.

20942 **int main(int argc, char \*argv[ ])**  
 20943 Calls *yylex()* to perform lexical analysis, then exits. The user code can contain *main()* to  
 20944 perform application-specific operations, calling *yylex()* as applicable.

20945 Except for *input()*, *unput()*, and *main()*, all external and static names generated by *lex* shall begin  
 20946 with the prefix **yy** or **YY**.

#### 20947 EXIT STATUS

20948 The following exit values shall be returned:

20949 0 Successful completion.

20950 >0 An error occurred.

#### 20951 CONSEQUENCES OF ERRORS

20952 Default.

#### 20953 APPLICATION USAGE

20954 Conforming applications are warned that in the *Rules* section, an ERE without an action is not  
 20955 acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or  
 20956 runtime errors.

20957 The purpose of *input()* is to take characters off the input stream and discard them as far as the  
 20958 lexical analysis is concerned. A common use is to discard the body of a comment once the  
 20959 beginning of a comment is recognized.

20960 The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex*  
 20961 source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer  
 20962 interpret the regular expressions given in the *lex* source according to the environment specified  
 20963 when the lexical analyzer is executed, but this is not possible with the current *lex* technology.  
 20964 Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the  
 20965 lexical requirements of the input language being described, which is frequently locale-specific  
 20966 anyway. (For example, writing an analyzer that is used for French text is not automatically  
 20967 useful for processing other languages.)

#### 20968 EXAMPLES

20969 The following is an example of a *lex* program that implements a rudimentary scanner for a  
 20970 Pascal-like syntax:

```
20971 %{
20972 /* Need this for the call to atof() below. */
20973 #include <math.h>
20974 /* Need this for printf(), fopen(), and stdin below. */
20975 #include <stdio.h>
20976 %}
```

```

20977 DIGIT [0-9]
20978 ID [a-z] [a-z0-9]*
20979
20980 %%
20981
20982 {DIGIT}+ {
20983 printf("An integer: %s (%d)\n", yytext,
20984 atoi(yytext));
20985 }
20986
20987 {DIGIT}+"."{DIGIT}* {
20988 printf("A float: %s (%g)\n", yytext,
20989 atof(yytext));
20990 }
20991
20992 if|then|begin|end|procedure|function {
20993 printf("A keyword: %s\n", yytext);
20994 }
20995
20996 {ID} printf("An identifier: %s\n", yytext);
20997
20998 "+"|"-"|"*"|"/" printf("An operator: %s\n", yytext);
20999
21000 "{"[^}\n]*}" /* Eat up one-line comments. */
21001
21002 [\t\n]+ /* Eat up white space. */
21003
21004 . printf("Unrecognized character: %s\n", yytext);
21005
21006 %%
21007
21008 int main(int argc, char *argv[])
21009 {
21010 ++argv, --argc; /* Skip over program name. */
21011 if (argc > 0)
21012 yyin = fopen(argv[0], "r");
21013 else
21014 yyin = stdin;
21015
21016 yylex();
21017 }

```

## 21006 RATIONALE

21007 Even though the `-c` option and references to the C language are retained in this description, *lex*  
 21008 may be generalized to other languages, as was done at one time for EFL, the Extended  
 21009 FORTRAN Language. Since the *lex* input specification is essentially language-independent,  
 21010 versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are  
 21011 known historical implementations that do so.

21012 The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex*  
 21013 source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is  
 21014 assumed to be presented in the POSIX locale, but input and output are in the locale specified by  
 21015 the environment variables), then the tables in the lexical analyzer produced by *lex* would  
 21016 interpret EREs specified in the *lex* source in terms of the environment variables specified when  
 21017 *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs  
 21018 given in the *lex* source according to the environment specified when the lexical analyzer is  
 21019 executed, but this is not possible with the current *lex* technology.

21020 The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard  
 21021 use of escape sequences. See the RATIONALE for *ed* for a discussion of bytes larger than 9 bits

21022 being represented by octal values. Hexadecimal values can represent larger bytes and multi-byte  
21023 characters directly, using as many digits as required.

21024 There is no detailed output format specification. The observed behavior of *lex* under four  
21025 different historical implementations was that none of these implementations consistently  
21026 reported the line numbers for error and warning messages. Furthermore, there was a desire that  
21027 *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified  
21028 avoids these formatting questions and problems with internationalization.

21029 Although the `%x` specifier for *exclusive* start conditions is not historical practice, it is believed to  
21030 be a minor change to historical implementations and greatly enhances the usability of *lex*  
21031 programs since it permits an application to obtain the expected functionality with fewer  
21032 statements.

21033 The `%array` and `%pointer` declarations were added as a compromise between historical systems.  
21034 The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in  
21035 BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements  
21036 are available for some scanners. Most historical programs should require no change in porting  
21037 from one system to another because the string being referenced is null-terminated in both cases.  
21038 (The method used by *flex* in its case is to null-terminate the token in place by remembering the  
21039 character that used to come right after the token and replacing it before continuing on to the next  
21040 scan.) Multi-file programs with external references to *yytext* outside the scanner source file  
21041 should continue to operate on their historical systems, but would require one of the new  
21042 declarations to be considered strictly portable.

21043 The description of EREs avoids unnecessary duplication of ERE details because their meanings  
21044 within a *lex* ERE are the same as that for the ERE in this volume of IEEE Std 1003.1-2001.

21045 The reason for the undefined condition associated with text beginning with a <blank> or within  
21046 "`%{`" and "`%}`" delimiter lines appearing in the *Rules* section is historical practice. Both the BSD  
21047 and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the  
21048 beginning) to unreachable areas of the *yylex()* function (the code is written directly after a *break*  
21049 statement). In some cases, the System V *lex* generates an error message or a syntax error,  
21050 depending on the form of indented input.

21051 The intention in breaking the list of functions into those that may appear in **lex.yy.c** versus those  
21052 that only appear in **libl.a** is that only those functions in **libl.a** can be reliably redefined by a  
21053 conforming application.

21054 The descriptions of standard output and standard error are somewhat complicated because  
21055 historical *lex* implementations chose to issue diagnostic messages to standard output (unless `-t`  
21056 was given). IEEE Std 1003.1-2001 allows this behavior, but leaves an opening for the more  
21057 expected behavior of using standard error for diagnostics. Also, the System V behavior of  
21058 writing the statistics when any table sizes are given is allowed, while BSD-derived systems can  
21059 avoid it. The programmer can always precisely obtain the desired results by using either the `-t`  
21060 or `-n` options.

21061 The OPERANDS section does not mention the use of `-` as a synonym for standard input; not all  
21062 historical implementations support such usage for any of the *file* operands.

21063 A description of the *translation table* was deleted from early proposals because of its relatively  
21064 low usage in historical applications.

21065 The change to the definition of the *input()* function that allows buffering of input presents the  
21066 opportunity for major performance gains in some applications.

21067 The following examples clarify the differences between *lex* regular expressions and regular  
21068 expressions appearing elsewhere in this volume of IEEE Std 1003.1-2001. For regular expressions

21069 of the form " $r/x$ ", the string matching  $r$  is always returned; confusion may arise when the  
21070 beginning of  $x$  matches the trailing portion of  $r$ . For example, given the regular expression  
21071 " $a*b/cc$ " and the input "aaabcc", *yytext* would contain the string "aaab" on this match. But  
21072 given the regular expression " $x*/xy$ " and the input "xxxxy", the token **xxx**, not **xx**, is returned  
21073 by some implementations because **xxx** matches " $x*$ ".

21074 In the rule " $ab*/bc$ ", the " $b*$ " at the end of  $r$  extends  $r$ 's match into the beginning of the  
21075 trailing context, so the result is unspecified. If this rule were " $ab/bc$ ", however, the rule  
21076 matches the text "ab" when it is followed by the text "bc". In this latter case, the matching of  $r$   
21077 cannot extend into the beginning of  $x$ , so the result is specified.

#### 21078 **FUTURE DIRECTIONS**

21079 None.

#### 21080 **SEE ALSO**

21081 *c99, ed, yacc*

#### 21082 **CHANGE HISTORY**

21083 First released in Issue 2.

#### 21084 **Issue 6**

21085 This utility is marked as part of the C-Language Development Utilities option.

21086 The obsolescent **-c** option is withdrawn in this issue.

21087 The normative text is reworded to avoid use of the term "must" for application requirements.

21088 **NAME**21089 link — call *link()* function21090 **SYNOPSIS**21091 xSI link *file1 file2*

21092

21093 **DESCRIPTION**21094 The *link* utility shall perform the function call:21095 link(*file1*, *file2*);21096 A user may need appropriate privilege to invoke the *link* utility.21097 **OPTIONS**

21098 None.

21099 **OPERANDS**

21100 The following operands shall be supported:

21101 *file1* The pathname of an existing file.21102 *file2* The pathname of the new directory entry to be created.21103 **STDIN**

21104 Not used.

21105 **INPUT FILES**

21106 Not used.

21107 **ENVIRONMENT VARIABLES**21108 The following environment variables shall affect the execution of *link*:

21109 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 21110 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 21111 Internationalization Variables for the precedence of internationalization variables  
 21112 used to determine the values of locale categories.)

21113 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 21114 internationalization variables.

21115 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 21116 characters (for example, single-byte as opposed to multi-byte characters in  
 21117 arguments).

21118 *LC\_MESSAGES*

21119 Determine the locale that should be used to affect the format and contents of  
 21120 diagnostic messages written to standard error.

21121 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.21122 **ASYNCHRONOUS EVENTS**

21123 Default.

21124 **STDOUT**

21125 None.

21126 **STDERR**

21127 The standard error shall be used only for diagnostic messages.

21128 **OUTPUT FILES**

21129 None.

21130 **EXTENDED DESCRIPTION**

21131 None.

21132 **EXIT STATUS**

21133 The following exit values shall be returned:

21134 0 Successful completion.

21135 &gt;0 An error occurred.

21136 **CONSEQUENCES OF ERRORS**

21137 Default.

21138 **APPLICATION USAGE**

21139 None.

21140 **EXAMPLES**

21141 None.

21142 **RATIONALE**

21143 None.

21144 **FUTURE DIRECTIONS**

21145 None.

21146 **SEE ALSO**21147 *ln*, *unlink*, the System Interfaces volume of IEEE Std 1003.1-2001, *link()*21148 **CHANGE HISTORY**

21149 First released in Issue 5.



21150 **NAME**

21151       ln — link files

21152 **SYNOPSIS**21153       ln [-fs] *source\_file target\_file*21154       ln [-fs] *source\_file ... target\_dir*21155 **DESCRIPTION**

21156       In the first synopsis form, the *ln* utility shall create a new directory entry (link) at the destination  
 21157       path specified by the *target\_file* operand. If the **-s** option is specified, a symbolic link shall be  
 21158       created for the file specified by the *source\_file* operand. This first synopsis form shall be assumed  
 21159       when the final operand does not name an existing directory; if more than two operands are  
 21160       specified and the final is not an existing directory, an error shall result.

21161       In the second synopsis form, the *ln* utility shall create a new directory entry (link), or if the **-s**  
 21162       option is specified a symbolic link, for each file specified by a *source\_file* operand, at a destination  
 21163       path in the existing directory named by *target\_dir*.

21164       If the last operand specifies an existing file of a type not specified by the System Interfaces  
 21165       volume of IEEE Std 1003.1-2001, the behavior is implementation-defined.

21166       The corresponding destination path for each *source\_file* shall be the concatenation of the target  
 21167       directory pathname, a slash character, and the last pathname component of the *source\_file*. The  
 21168       second synopsis form shall be assumed when the final operand names an existing directory.

21169       For each *source\_file*:

- 21170       1. If the destination path exists:
  - 21171           a. If the **-f** option is not specified, *ln* shall write a diagnostic message to standard error,  
 21172           do nothing more with the current *source\_file*, and go on to any remaining *source\_files*.
  - 21173           b. Actions shall be performed equivalent to the *unlink()* function defined in the System  
 21174           Interfaces volume of IEEE Std 1003.1-2001, called using *destination* as the *path*  
 21175           argument. If this fails for any reason, *ln* shall write a diagnostic message to standard  
 21176           error, do nothing more with the current *source\_file*, and go on to any remaining  
 21177           *source\_files*.
- 21178       2. If the **-s** option is specified, *ln* shall create a symbolic link named by the destination path  
 21179       and containing as its pathname *source\_file*. The *ln* utility shall do nothing more with  
 21180       *source\_file* and shall go on to any remaining files.
- 21181       3. If *source\_file* is a symbolic link, actions shall be performed equivalent to the *link()* function  
 21182       using the object that *source\_file* references as the *path1* argument and the destination path  
 21183       as the *path2* argument. The *ln* utility shall do nothing more with *source\_file* and shall go on  
 21184       to any remaining files.
- 21185       4. Actions shall be performed equivalent to the *link()* function defined in the System  
 21186       Interfaces volume of IEEE Std 1003.1-2001 using *source\_file* as the *path1* argument, and the  
 21187       destination path as the *path2* argument.

21188 **OPTIONS**

21189       The *ln* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 21190       Utility Syntax Guidelines.

21191       The following option shall be supported:

21192       **-f**           Force existing destination pathnames to be removed to allow the link.

21193            **-s**            Create symbolic links instead of hard links.

#### 21194 **OPERANDS**

21195            The following operands shall be supported:

21196            *source\_file*      A pathname of a file to be linked. If the **-s** option is specified, no restrictions on the  
21197                                   type of file or on its existence shall be made. If the **-s** option is not specified,  
21198                                   whether a directory can be linked is implementation-defined.

21199            *target\_file*        The pathname of the new directory entry to be created.

21200            *target\_dir*          A pathname of an existing directory in which the new directory entries are created.

#### 21201 **STDIN**

21202            Not used.

#### 21203 **INPUT FILES**

21204            None.

#### 21205 **ENVIRONMENT VARIABLES**

21206            The following environment variables shall affect the execution of *ln*:

21207            *LANG*                Provide a default value for the internationalization variables that are unset or null.  
21208                                   (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
21209                                   Internationalization Variables for the precedence of internationalization variables  
21210                                   used to determine the values of locale categories.)

21211            *LC\_ALL*              If set to a non-empty string value, override the values of all the other  
21212                                   internationalization variables.

21213            *LC\_CTYPE*        Determine the locale for the interpretation of sequences of bytes of text data as  
21214                                   characters (for example, single-byte as opposed to multi-byte characters in  
21215                                   arguments).

21216            *LC\_MESSAGES*

21217                                   Determine the locale that should be used to affect the format and contents of  
21218                                   diagnostic messages written to standard error.

21219 xSI        *NLSPATH*        Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 21220 **ASYNCHRONOUS EVENTS**

21221            Default.

#### 21222 **STDOUT**

21223            Not used.

#### 21224 **STDERR**

21225            The standard error shall be used only for diagnostic messages.

#### 21226 **OUTPUT FILES**

21227            None.

#### 21228 **EXTENDED DESCRIPTION**

21229            None.

#### 21230 **EXIT STATUS**

21231            The following exit values shall be returned:

21232            0    All the specified files were linked successfully.

21233            >0   An error occurred.

21234 **CONSEQUENCES OF ERRORS**

21235 Default.

21236 **APPLICATION USAGE**

21237 None.

21238 **EXAMPLES**

21239 None.

21240 **RATIONALE**

21241 Some historic versions of *ln* (including the one specified by the SVID) unlink the destination file,  
 21242 if it exists, by default. If the mode does not permit writing, these versions prompt for  
 21243 confirmation before attempting the unlink. In these versions the **-f** option causes *ln* not to  
 21244 attempt to prompt for confirmation.

21245 This allows *ln* to succeed in creating links when the target file already exists, even if the file itself  
 21246 is not writable (although the directory must be). Early proposals specified this functionality.

21247 This volume of IEEE Std 1003.1-2001 does not allow the *ln* utility to unlink existing destination  
 21248 paths by default for the following reasons:

- 21249 • The *ln* utility has historically been used to provide locking for shell applications, a usage that  
 21250 is incompatible with *ln* unlinking the destination path by default. There was no  
 21251 corresponding technical advantage to adding this functionality.
- 21252 • This functionality gave *ln* the ability to destroy the link structure of files, which changes the  
 21253 historical behavior of *ln*.
- 21254 • This functionality is easily replicated with a combination of *rm* and *ln*.
- 21255 • It is not historical practice in many systems; BSD and BSD-derived systems do not support  
 21256 this behavior. Unfortunately, whichever behavior is selected can cause scripts written  
 21257 expecting the other behavior to fail.
- 21258 • It is preferable that *ln* perform in the same manner as the *link()* function, which does not  
 21259 permit the target to exist already.

21260 This volume of IEEE Std 1003.1-2001 retains the **-f** option to provide support for shell scripts  
 21261 depending on the SVID semantics. It seems likely that shell scripts would not be written to  
 21262 handle prompting by *ln* and would therefore have specified the **-f** option.

21263 The **-f** option is an undocumented feature of many historical versions of the *ln* utility, allowing  
 21264 linking to directories. These versions require modification.

21265 Early proposals of this volume of IEEE Std 1003.1-2001 also required a **-i** option, which behaved  
 21266 like the **-i** options in *cp* and *mv*, prompting for confirmation before unlinking existing files. This  
 21267 was not historical practice for the *ln* utility and has been omitted.

21268 **FUTURE DIRECTIONS**

21269 None.

21270 **SEE ALSO**21271 *chmod*, *find*, *pax*, *rm*, the System Interfaces volume of IEEE Std 1003.1-2001, *link()*, *unlink()*21272 **CHANGE HISTORY**

21273 First released in Issue 2.

21274 **Issue 6**

21275 The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b  
21276 draft standard.

21277 **NAME**

21278           locale — get locale-specific information

21279 **SYNOPSIS**

21280           locale [-a | -m]

21281           locale [-ck] *name*...21282 **DESCRIPTION**

21283           The *locale* utility shall write information about the current locale environment, or all public  
 21284           locales, to the standard output. For the purposes of this section, a *public locale* is one provided by  
 21285           the implementation that is accessible to the application.

21286           When *locale* is invoked without any arguments, it shall summarize the current locale  
 21287           environment for each locale category as determined by the settings of the environment variables  
 21288           defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21289           When invoked with operands, it shall write values that have been assigned to the keywords in  
 21290           the locale categories, as follows:

- 21291           • Specifying a keyword name shall select the named keyword and the category containing that  
 21292           keyword.
- 21293           • Specifying a category name shall select the named category and all keywords in that  
 21294           category.

21295 **OPTIONS**

21296           The *locale* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 21297           12.2, Utility Syntax Guidelines.

21298           The following options shall be supported:

- 21299           **-a**           Write information about all available public locales. The available locales shall  
 21300           include **POSIX**, representing the POSIX locale. The manner in which the  
 21301           implementation determines what other locales are available is implementation-  
 21302           defined.
- 21303           **-c**           Write the names of selected locale categories; see the STDOUT section. The **-c**  
 21304           option increases readability when more than one category is selected (for example,  
 21305           via more than one keyword name or via a category name). It is valid both with  
 21306           and without the **-k** option.
- 21307           **-k**           Write the names and values of selected keywords. The implementation may omit  
 21308           values for some keywords; see the OPERANDS section.
- 21309           **-m**           Write names of available charmaps; see the Base Definitions volume of  
 21310           IEEE Std 1003.1-2001, Section 6.1, Portable Character Set.

21311 **OPERANDS**

21312           The following operand shall be supported:

- 21313           *name*       The name of a locale category as defined in the Base Definitions volume of  
 21314           IEEE Std 1003.1-2001, Chapter 7, Locale, the name of a keyword in a locale  
 21315           category, or the reserved name **charmap**. The named category or keyword shall be  
 21316           selected for output. If a single *name* represents both a locale category name and a  
 21317           keyword name in the current locale, the results are unspecified. Otherwise, both  
 21318           category and keyword names can be specified as *name* operands, in any sequence.  
 21319           It is implementation-defined whether any keyword values are written for the  
 21320           categories *LC\_CTYPE* and *LC\_COLLATE*.

21321 **STDIN**

21322 Not used.

21323 **INPUT FILES**

21324 None.

21325 **ENVIRONMENT VARIABLES**21326 The following environment variables shall affect the execution of *locale*:

21327 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 21328 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 21329 Internationalization Variables for the precedence of internationalization variables  
 21330 used to determine the values of locale categories.)

21331 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 21332 internationalization variables.

21333 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 21334 characters (for example, single-byte as opposed to multi-byte characters in  
 21335 arguments and input files).

21336 *LC\_MESSAGES*

21337 Determine the locale that should be used to affect the format and contents of  
 21338 diagnostic messages written to standard error.

21339 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

21340 XSI The application shall ensure that the *LANG*, *LC\_\**, and *NLSPATH* environment variables specify  
 21341 the current locale environment to be written out; they shall be used if the *-a* option is not  
 21342 specified.

21343 **ASYNCHRONOUS EVENTS**

21344 Default.

21345 **STDOUT**

21346 If *locale* is invoked without any options or operands, the names and values of the *LANG* and  
 21347 *LC\_\** environment variables described in this volume of IEEE Std 1003.1-2001 shall be written to  
 21348 the standard output, one variable per line, with *LANG* first, and each line using the following  
 21349 format. Only those variables set in the environment and not overridden by *LC\_ALL* shall be  
 21350 written using this format:

21351 "%s=%s\n", &lt;variable\_name&gt;, &lt;value&gt;

21352 The names of those *LC\_\** variables associated with locale categories defined in this volume of  
 21353 IEEE Std 1003.1-2001 that are not set in the environment or are overridden by *LC\_ALL* shall be  
 21354 written in the following format:

21355 "%s=\"%s\"\n", &lt;variable\_name&gt;, &lt;implied value&gt;

21356 The <implied value> shall be the name of the locale that has been selected for that category by the  
 21357 implementation, based on the values in *LANG* and *LC\_ALL*, as described in the Base Definitions  
 21358 volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

21359 The <value> and <implied value> shown above shall be properly quoted for possible later reentry  
 21360 to the shell. The <value> shall not be quoted using double-quotes (so that it can be distinguished  
 21361 by the user from the <implied value> case, which always requires double-quotes).

21362 The *LC\_ALL* variable shall be written last, using the first format shown above. If it is not set, it  
 21363 shall be written as:

21364 "LC\_ALL=\n"

21365 If any arguments are specified:

21366 1. If the **-a** option is specified, the names of all the public locales shall be written, each in the  
21367 following format:

21368 "%s\n", <locale name>

21369 2. If the **-c** option is specified, the names of all selected categories shall be written, each in the  
21370 following format:

21371 "%s\n", <category name>

21372 If keywords are also selected for writing (see following items), the category name output  
21373 shall precede the keyword output for that category.

21374 If the **-c** option is not specified, the names of the categories shall not be written; only the  
21375 keywords, as selected by the <name> operand, shall be written.

21376 3. If the **-k** option is specified, the names and values of selected keywords shall be written. If  
21377 a value is non-numeric, it shall be written in the following format:

21378 "%s=\"%s\"\\n", <keyword name>, <keyword value>

21379 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the  
21380 *localedef* **-f** option when the locale was created shall be written, with the word **charmap** as  
21381 <keyword name>.

21382 If a value is numeric, it shall be written in one of the following formats:

21383 "%s=%d\n", <keyword name>, <keyword value>

21384 "%s=%c%o\n", <keyword name>, <escape character>, <keyword value>

21385 "%s=%cx%x\n", <keyword name>, <escape character>, <keyword value>

21386 where the <escape character> is that identified by the **escape\_char** keyword in the current  
21387 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3, Locale  
21388 Definition.

21389 Compound keyword values (list entries) shall be separated in the output by semicolons.  
21390 When included in keyword values, the semicolon, the double-quote, the backslash, and  
21391 any control character shall be preceded (escaped) with the escape character.

21392 4. If the **-k** option is not specified, selected keyword values shall be written, each in the  
21393 following format:

21394 "%s\n", <keyword value>

21395 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the  
21396 *localedef* **-f** option when the locale was created shall be written.

21397 5. If the **-m** option is specified, then a list of all available charmaps shall be written, each in  
21398 the format:

21399 "%s\n", <charmap>

21400 where <charmap> is in a format suitable for use as the option-argument to the *localedef* **-f**  
21401 option.

21402 **STDERR**

21403       The standard error shall be used only for diagnostic messages.

21404 **OUTPUT FILES**

21405       None.

21406 **EXTENDED DESCRIPTION**

21407       None.

21408 **EXIT STATUS**

21409       The following exit values shall be returned:

21410       0   All the requested information was found and output successfully.

21411       >0   An error occurred.

21412 **CONSEQUENCES OF ERRORS**

21413       Default.

21414 **APPLICATION USAGE**

21415       If the *LANG* environment variable is not set or set to an empty value, or one of the *LC\_\**  
21416       environment variables is set to an unrecognized value, the actual locales assumed (if any) are  
21417       implementation-defined as described in the Base Definitions volume of IEEE Std 1003.1-2001,  
21418       Chapter 8, Environment Variables.

21419       Implementations are not required to write out the actual values for keywords in the categories  
21420       *LC\_CTYPE* and *LC\_COLLATE*; however, they must write out the categories (allowing an  
21421       application to determine, for example, which character classes are available).

21422 **EXAMPLES**

21423       In the following examples, the assumption is that locale environment variables are set as  
21424       follows:

21425       LANG=locale\_x  
21426       LC\_COLLATE=locale\_y

21427       The command *locale* would result in the following output:

21428       LANG=locale\_x  
21429       LC\_CTYPE="locale\_x"  
21430       LC\_COLLATE=locale\_y  
21431       LC\_TIME="locale\_x"  
21432       LC\_NUMERIC="locale\_x"  
21433       LC\_MONETARY="locale\_x"  
21434       LC\_MESSAGES="locale\_x"  
21435       LC\_ALL=

21436       The order of presentation of the categories is not specified by this volume of  
21437       IEEE Std 1003.1-2001.

21438       The command:

21439       LC\_ALL=POSIX locale -ck decimal\_point

21440       would produce:

21441       LC\_NUMERIC  
21442       decimal\_point="."

21443       The following command shows an application of *locale* to determine whether a user-supplied  
21444       response is affirmative:



```

21445 if printf "%s\n" "$response" | grep -Eq "$(locale yesexpr) "
21446 then
21447 affirmative processing goes here
21448 else
21449 non-affirmative processing goes here
21450 fi

```

#### 21451 **RATIONALE**

21452 The output for categories *LC\_CTYPE* and *LC\_COLLATE* has been made implementation-defined  
 21453 because there is a questionable value in having a shell script receive an entire array of characters.  
 21454 It is also difficult to return a logical collation description, short of returning a complete *localedef*  
 21455 source.

21456 The **-m** option was included to allow applications to query for the existence of charmaps. The  
 21457 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the  
 21458 system.

21459 The **-c** option was included for readability when more than one category is selected (for  
 21460 example, via more than one keyword name or via a category name). It is valid both with and  
 21461 without the **-k** option.

21462 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the  
 21463 current locale was created, was included to allow applications needing the information to  
 21464 retrieve it.

#### 21465 **FUTURE DIRECTIONS**

21466 None.

#### 21467 **SEE ALSO**

21468 *localedef*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3, Locale Definition

#### 21469 **CHANGE HISTORY**

21470 First released in Issue 4.

#### 21471 **Issue 5**

21472 The FUTURE DIRECTIONS section is added.

#### 21473 **Issue 6**

21474 The normative text is reworded to avoid use of the term “must” for application requirements.

21475 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/30 is applied, correcting an editorial error |  
 21476 in the STDOUT section. |

21477 **NAME**

21478 localedef — define locale environment

21479 **SYNOPSIS**

21480 localedef [-c] [-f *charmap*] [-i *sourcefile*] [-u *code\_set\_name*] *name*

21481 **DESCRIPTION**

21482 The *localedef* utility shall convert source definitions for locale categories into a format usable by  
 21483 the functions and utilities whose operational behavior is determined by the setting of the locale  
 21484 environment variables defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter  
 21485 7, Locale. It is implementation-defined whether users have the capability to create new locales,  
 21486 in addition to those supplied by the implementation. If the symbolic constant  
 21487 XSI POSIX2\_LOCALEDEF is defined, the system supports the creation of new locales. On XSI-  
 21488 conformant systems, the symbolic constant POSIX2\_LOCALEDEF shall be defined.

21489 The utility shall read source definitions for one or more locale categories belonging to the same  
 21490 locale from the file named in the -i option (if specified) or from standard input.

21491 The *name* operand identifies the target locale. The utility shall support the creation of *public*, or  
 21492 generally accessible locales, as well as *private*, or restricted-access locales. Implementations may  
 21493 restrict the capability to create or modify public locales to users with the appropriate privileges.

21494 Each category source definition shall be identified by the corresponding environment variable  
 21495 name and terminated by an **END** *category-name* statement. The following categories shall be  
 21496 supported. In addition, the input may contain source for implementation-defined categories.

21497 *LC\_CTYPE* Defines character classification and case conversion.

21498 *LC\_COLLATE*  
 21499 Defines collation rules.

21500 *LC\_MONETARY*  
 21501 Defines the format and symbols used in formatting of monetary information.

21502 *LC\_NUMERIC*  
 21503 Defines the decimal delimiter, grouping, and grouping symbol for non-monetary  
 21504 numeric editing.

21505 *LC\_TIME* Defines the format and content of date and time information.

21506 *LC\_MESSAGES*  
 21507 Defines the format and values of affirmative and negative responses.

21508 **OPTIONS**

21509 The *localedef* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 21510 12.2, Utility Syntax Guidelines.

21511 The following options shall be supported:

21512 -c Create permanent output even if warning messages have been issued.

21513 -f *charmap* Specify the pathname of a file containing a mapping of character symbols and  
 21514 collating element symbols to actual character encodings. The format of the  
 21515 *charmap* is described in the Base Definitions volume of IEEE Std 1003.1-2001,  
 21516 Section 6.4, Character Set Description File. The application shall ensure that this  
 21517 option is specified if symbolic names (other than collating symbols defined in a  
 21518 **collating-symbol** keyword) are used. If the -f option is not present, an  
 21519 implementation-defined character mapping shall be used.

21520        **-i** *inputfile*   The pathname of a file containing the source definitions. If this option is not  
 21521                           present, source definitions shall be read from standard input. The format of the  
 21522                           *inputfile* is described in the Base Definitions volume of IEEE Std 1003.1-2001,  
 21523                           Section 7.3, Locale Definition.

21524        **-u** *code\_set\_name*       Specify the name of a codeset used as the target mapping of character symbols and  
 21525                           collating element symbols whose encoding values are defined in terms of the  
 21526                           ISO/IEC 10646-1:2000 standard position constant values.  
 21527

## 21528 OPERANDS

21529        The following operand shall be supported:

21530        *name*           Identifies the locale; see the Base Definitions volume of IEEE Std 1003.1-2001,  
 21531                           Chapter 7, Locale for a description of the use of this name. If the name contains one  
 21532                           or more slash characters, *name* shall be interpreted as a pathname where the  
 21533                           created locale definitions shall be stored. If *name* does not contain any slash  
 21534                           characters, the interpretation of the name is implementation-defined and the locale  
 21535                           shall be public. This capability may be restricted to users with appropriate  
 21536                           privileges. (As a consequence of specifying one *name*, although several categories  
 21537                           can be processed in one execution, only categories belonging to the same locale can  
 21538                           be processed.)

## 21539 STDIN

21540        Unless the **-i** option is specified, the standard input shall be a text file containing one or more  
 21541        locale category source definitions, as described in the Base Definitions volume of  
 21542        IEEE Std 1003.1-2001, Section 7.3, Locale Definition. When lines are continued using the escape  
 21543        character mechanism, there is no limit to the length of the accumulated continued line.

## 21544 INPUT FILES

21545        The character set mapping file specified as the *charmap* option-argument is described in the Base  
 21546        Definitions volume of IEEE Std 1003.1-2001, Section 6.4, Character Set Description File. If a locale  
 21547        category source definition contains a **copy** statement, as defined in the Base Definitions volume  
 21548        of IEEE Std 1003.1-2001, Chapter 7, Locale, and the **copy** statement names a valid, existing locale,  
 21549        then *localedef* shall behave as if the source definition had contained a valid category source  
 21550        definition for the named locale.

## 21551 ENVIRONMENT VARIABLES

21552        The following environment variables shall affect the execution of *localedef*:

21553        **LANG**           Provide a default value for the internationalization variables that are unset or null.  
 21554                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 21555                           Internationalization Variables for the precedence of internationalization variables  
 21556                           used to determine the values of locale categories.)

21557        **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
 21558                           internationalization variables.

21559        **LC\_COLLATE**        (This variable has no affect on *localedef*; the POSIX locale is used for this category.)  
 21560

21561        **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
 21562                           characters (for example, single-byte as opposed to multi-byte characters in  
 21563                           arguments and input files). This variable has no affect on the processing of *localedef*  
 21564                           input data; the POSIX locale is used for this purpose, regardless of the value of this  
 21565                           variable.

21566 *LC\_MESSAGES*

21567 Determine the locale that should be used to affect the format and contents of  
21568 diagnostic messages written to standard error.

21569 xSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 21570 ASYNCHRONOUS EVENTS

21571 Default.

## 21572 STDOUT

21573 The utility shall report all categories successfully processed, in an unspecified format.

## 21574 STDERR

21575 The standard error shall be used only for diagnostic messages.

## 21576 OUTPUT FILES

21577 The format of the created output is unspecified. If the *name* operand does not contain a slash, the  
21578 existence of an output file for the locale is unspecified.

## 21579 EXTENDED DESCRIPTION

21580 When the *-u* option is used, the *code\_set\_name* option-argument shall be interpreted as an  
21581 implementation-defined name of a codeset to which the ISO/IEC 10646-1:2000 standard  
21582 position constant values shall be converted via an implementation-defined method. Both the  
21583 ISO/IEC 10646-1:2000 standard position constant values and other formats (decimal,  
21584 hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset  
21585 represented by the implementation-defined name can be any codeset that is supported by the  
21586 implementation.

21587 When conflicts occur between the *charmap* specification of *<code\_set\_name>*, *<mb\_cur\_max>*, or  
21588 *<mb\_cur\_min>* and the implementation-defined interpretation of these respective items for the  
21589 codeset represented by the *-u* option-argument *code\_set\_name*, the result is unspecified.

21590 When conflicts occur between the *charmap* encoding values specified for symbolic names of  
21591 characters of the portable character set and the implementation-defined assignment of character  
21592 encoding values, the result is unspecified.

21593 If a non-printable character in the *charmap* has a width specified that is not *-1*, *localedef* shall  
21594 generate a warning.

## 21595 EXIT STATUS

21596 The following exit values shall be returned:

- 21597 0 No errors occurred and the locales were successfully created.
- 21598 1 Warnings occurred and the locales were successfully created.
- 21599 2 The locale specification exceeded implementation limits or the coded character set or sets  
21600 used were not supported by the implementation, and no locale was created.
- 21601 3 The capability to create new locales is not supported by the implementation.
- 21602 >3 Warnings or errors occurred and no output was created.

## 21603 CONSEQUENCES OF ERRORS

21604 If an error is detected, no permanent output shall be created.

21605 If warnings occur, permanent output shall be created if the *-c* option was specified. The  
21606 following conditions shall cause warning messages to be issued:

- 21607 • If a symbolic name not found in the *charmap* file is used for the descriptions of the *LC\_CTYPE*  
21608 or *LC\_COLLATE* categories (for other categories, this shall be an error condition).

- 21609           • If the number of operands to the **order** keyword exceeds the {COLL\_WEIGHTS\_MAX} limit.
- 21610           • If optional keywords not supported by the implementation are present in the source.
- 21611           • If a non-printable character has a width specified other than -1.
- 21612           Other implementation-defined conditions may also cause warnings.

#### 21613 **APPLICATION USAGE**

21614           The *charmap* definition is optional, and is contained outside the locale definition. This allows  
21615           both completely self-defined source files, and generic sources (applicable to more than one  
21616           codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the  
21617           portable character set. As explained in the Base Definitions volume of IEEE Std 1003.1-2001,  
21618           Section 6.4, Character Set Description File, it is implementation-defined whether or not users or  
21619           applications can provide additional character set description files. Therefore, the **-f** option might  
21620           be operable only when an implementation-defined *charmap* is named.

#### 21621 **EXAMPLES**

21622           None.

#### 21623 **RATIONALE**

21624           The output produced by the *localedef* utility is implementation-defined. The *name* operand is  
21625           used to identify the specific locale. (As a consequence, although several categories can be  
21626           processed in one execution, only categories belonging to the same locale can be processed.)

#### 21627 **FUTURE DIRECTIONS**

21628           None.

#### 21629 **SEE ALSO**

21630           *locale*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3, Locale Definition

#### 21631 **CHANGE HISTORY**

21632           First released in Issue 4.

#### 21633 **Issue 6**

21634           The **-u** option is added, as specified in the IEEE P1003.2b draft standard.

21635           The normative text is reworded to avoid use of the term “must” for application requirements.

21636 **NAME**

21637           logger — log messages

21638 **SYNOPSIS**21639           logger *string* ...21640 **DESCRIPTION**

21641           The *logger* utility saves a message, in an unspecified manner and format, containing the *string*  
 21642           operands provided by the user. The messages are expected to be evaluated later by personnel  
 21643           performing system administration tasks.

21644           It is implementation-defined whether messages written in locales other than the POSIX locale  
 21645           are effective.

21646 **OPTIONS**

21647           None.

21648 **OPERANDS**

21649           The following operand shall be supported:

21650           *string*       One of the string arguments whose contents are concatenated together, in the  
 21651                           order specified, separated by single <space>s.

21652 **STDIN**

21653           Not used.

21654 **INPUT FILES**

21655           None.

21656 **ENVIRONMENT VARIABLES**21657           The following environment variables shall affect the execution of *logger*:

21658           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 21659                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 21660                           Internationalization Variables for the precedence of internationalization variables  
 21661                           used to determine the values of locale categories.)

21662           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 21663                           internationalization variables.

21664           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 21665                           characters (for example, single-byte as opposed to multi-byte characters in  
 21666                           arguments).

21667           *LC\_MESSAGES*

21668                           Determine the locale that should be used to affect the format and contents of  
 21669                           diagnostic messages written to standard error. (This means diagnostics from *logger*  
 21670                           to the user or application, not diagnostic messages that the user is sending to the  
 21671                           system administrator.)

21672 *XSI*       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

21673 **ASYNCHRONOUS EVENTS**

21674           Default.

21675 **STDOUT**

21676           Not used.

21677 **STDERR**

21678           The standard error shall be used only for diagnostic messages.

21679 **OUTPUT FILES**

21680           Unspecified.

21681 **EXTENDED DESCRIPTION**

21682           None.

21683 **EXIT STATUS**

21684           The following exit values shall be returned:

21685           0   Successful completion.

21686           >0  An error occurred.

21687 **CONSEQUENCES OF ERRORS**

21688           Default.

21689 **APPLICATION USAGE**

21690           This utility allows logging of information for later use by a system administrator or programmer  
21691           in determining why non-interactive utilities have failed. The locations of the saved messages,  
21692           their format, and retention period are all unspecified. There is no method for a conforming  
21693           application to read messages, once written.

21694 **EXAMPLES**

21695           A batch application, running non-interactively, tries to read a configuration file and fails; it may  
21696           attempt to notify the system administrator with:

21697           logger myname: unable to read file foo. [timestamp]

21698 **RATIONALE**

21699           The standard developers believed strongly that some method of alerting administrators to errors  
21700           was necessary. The obvious example is a batch utility, running non-interactively, that is unable  
21701           to read its configuration files or that is unable to create or write its results file. However, the  
21702           standard developers did not wish to define the format or delivery mechanisms as they have  
21703           historically been (and will probably continue to be) very system-specific, as well as involving  
21704           functionality clearly outside the scope of this volume of IEEE Std 1003.1-2001.

21705           The text with *LC\_MESSAGES* about diagnostic messages means diagnostics from *logger* to the  
21706           user or application, not diagnostic messages that the user is sending to the system administrator.

21707           Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

21708           Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient  
21709           justification to exclude these utilities from this volume of IEEE Std 1003.1-2001. It is also  
21710           arguable that they are, in fact, testable, but that the tests themselves are not portable.

21711 **FUTURE DIRECTIONS**

21712           None.

21713 **SEE ALSO**

21714           *lp*, *mailx*, *write*

21715 **CHANGE HISTORY**

21716           First released in Issue 4.

21717 **NAME**

21718 logname — return the user's login name

21719 **SYNOPSIS**

21720 logname

21721 **DESCRIPTION**

21722 The *logname* utility shall write the user's login name to standard output. The login name shall be  
 21723 the string that would be returned by the *getlogin()* function defined in the System Interfaces  
 21724 volume of IEEE Std 1003.1-2001. Under the conditions where the *getlogin()* function would fail,  
 21725 the *logname* utility shall write a diagnostic message to standard error and exit with a non-zero  
 21726 exit status.

21727 **OPTIONS**

21728 None.

21729 **OPERANDS**

21730 None.

21731 **STDIN**

21732 Not used.

21733 **INPUT FILES**

21734 None.

21735 **ENVIRONMENT VARIABLES**21736 The following environment variables shall affect the execution of *logname*:

21737 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 21738 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 21739 Internationalization Variables for the precedence of internationalization variables  
 21740 used to determine the values of locale categories.)

21741 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 21742 internationalization variables.

21743 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 21744 characters (for example, single-byte as opposed to multi-byte characters in  
 21745 arguments).

21746 *LC\_MESSAGES*

21747 Determine the locale that should be used to affect the format and contents of  
 21748 diagnostic messages written to standard error.

21749 *XS1* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

21750 **ASYNCHRONOUS EVENTS**

21751 Default.

21752 **STDOUT**21753 The *logname* utility output shall be a single line consisting of the user's login name:

21754 "%s\n", &lt;login name&gt;

21755 **STDERR**

21756 The standard error shall be used only for diagnostic messages.



21757 **OUTPUT FILES**

21758           None.

21759 **EXTENDED DESCRIPTION**

21760           None.

21761 **EXIT STATUS**

21762           The following exit values shall be returned:

21763           0   Successful completion.

21764           &gt;0  An error occurred.

21765 **CONSEQUENCES OF ERRORS**

21766           Default.

21767 **APPLICATION USAGE**21768           The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment changes could produce erroneous results.21770 **EXAMPLES**

21771           None.

21772 **RATIONALE**21773           The **passwd** file is not listed as required because the implementation may have other means of mapping login names.21775 **FUTURE DIRECTIONS**

21776           None.

21777 **SEE ALSO**21778           *id*, *who*, the System Interfaces volume of IEEE Std 1003.1-2001, *getlogin()*21779 **CHANGE HISTORY**

21780           First released in Issue 2.

21781 **NAME**

21782       lp — send files to a printer

21783 **SYNOPSIS**21784       lp [-c] [-d *dest*] [-n *copies*] [-msw] [-o *option*]... [-t *title*] [*file*...]21785 **DESCRIPTION**

21786       The *lp* utility shall copy the input files to an output destination in an unspecified manner. The  
 21787       default output destination should be to a hardcopy device, such as a printer or microfilm  
 21788       recorder, that produces non-volatile, human-readable documents. If such a device is not  
 21789       available to the application, or if the system provides no such device, the *lp* utility shall exit with  
 21790       a non-zero exit status.

21791       The actual writing to the output device may occur some time after the *lp* utility successfully  
 21792       exits. During the portion of the writing that corresponds to each input file, the implementation  
 21793       shall guarantee exclusive access to the device.

21794       The *lp* utility shall associate a unique *request ID* with each request.

21795       Normally, a banner page is produced to separate and identify each print job. This page may be  
 21796       suppressed by implementation-defined conditions, such as an operator command or one of the  
 21797       -o *option* values.

21798 **OPTIONS**

21799       The *lp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 21800       Utility Syntax Guidelines.

21801       The following options shall be supported:

21802       -c       Exit only after further access to any of the input files is no longer required. The  
 21803       application can then safely delete or modify the files without affecting the output  
 21804       operation. Normally, files are not copied, but are linked whenever possible. If the  
 21805       -c option is not given, then the user should be careful not to remove any of the  
 21806       files before the request has been printed in its entirety. It should also be noted that  
 21807       in the absence of the -c option, any changes made to the named files after the  
 21808       request is made but before it is printed may be reflected in the printed output. On  
 21809       some implementations, -c may be on by default.

21810       -d *dest*   Specify a string that names the destination (*dest*). If *dest* is a printer, the request  
 21811       shall be printed only on that specific printer. If *dest* is a class of printers, the request  
 21812       shall be printed on the first available printer that is a member of the class. Under  
 21813       certain conditions (printer unavailability, file space limitation, and so on), requests  
 21814       for specific destinations need not be accepted. Destination names vary between  
 21815       systems.

21816       If -d is not specified, and neither the *LPDEST* nor *PRINTER* environment variable  
 21817       is set, an unspecified destination is used. The -d *dest* option shall take precedence  
 21818       over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are  
 21819       undefined when *dest* contains a value that is not a valid destination name.

21820       -m       Send mail (see *mailx*) after the files have been printed. By default, no mail is sent  
 21821       upon normal completion of the print request.

21822       -n *copies*   Write *copies* number of copies of the files, where *copies* is a positive decimal integer.  
 21823       The methods for producing multiple copies and for arranging the multiple copies  
 21824       when multiple *file* operands are used are unspecified, except that each file shall be  
 21825       output as an integral whole, not interleaved with portions of other files.

21826        **-o option**        Specify printer-dependent or class-dependent *options*. Several such *options* may be  
 21827                                collected by specifying the **-o** option more than once.

21828        **-s**                Suppress messages from *lp*.

21829        **-t title**        Write *title* on the banner page of the output.

21830        **-w**                Write a message on the user's terminal after the files have been printed. If the user  
 21831                                is not logged in, then mail shall be sent instead.

## 21832 OPERANDS

21833        The following operand shall be supported:

21834        *file*              A pathname of a file to be output. If no *file* operands are specified, or if a *file*  
 21835                                operand is '-', the standard input shall be used. If a *file* operand is used, but the  
 21836                                **-c** option is not specified, the process performing the writing to the output device  
 21837                                may have user and group permissions that differ from that of the process invoking  
 21838                                *lp*.

## 21839 STDIN

21840        The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
 21841        See the INPUT FILES section.

## 21842 INPUT FILES

21843        The input files shall be text files.

## 21844 ENVIRONMENT VARIABLES

21845        The following environment variables shall affect the execution of *lp*:

21846        *LANG*              Provide a default value for the internationalization variables that are unset or null.  
 21847                                (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 21848                                Internationalization Variables for the precedence of internationalization variables  
 21849                                used to determine the values of locale categories.)

21850        *LC\_ALL*            If set to a non-empty string value, override the values of all the other  
 21851                                internationalization variables.

21852        *LC\_CTYPE*        Determine the locale for the interpretation of sequences of bytes of text data as  
 21853                                characters (for example, single-byte as opposed to multi-byte characters in  
 21854                                arguments and input files).

### 21855 LC\_MESSAGES

21856                                Determine the locale that should be used to affect the format and contents of  
 21857                                diagnostic messages written to standard error and informative messages written to  
 21858                                standard output.

21859        *LC\_TIME*        Determine the format and contents of date and time strings displayed in the *lp*  
 21860                                banner page, if any.

21861        *LPDEST*            Determine the destination. If the *LPDEST* environment variable is not set, the  
 21862                                *PRINTER* environment variable shall be used. The **-d dest** option takes precedence  
 21863                                over *LPDEST*. Results are undefined when **-d** is not specified and *LPDEST*  
 21864                                contains a value that is not a valid destination name.

21865 xSI        *NLS\_PATH*        Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

21866        *PRINTER*        Determine the output device or destination. If the *LPDEST* and *PRINTER*  
 21867                                environment variables are not set, an unspecified output device is used. The **-d**  
 21868                                *dest* option and the *LPDEST* environment variable shall take precedence over  
 21869                                *PRINTER*. Results are undefined when **-d** is not specified, *LPDEST* is unset, and

21870 *PRINTER* contains a value that is not a valid device or destination name.

21871 *TZ* Determine the timezone used to calculate date and time strings displayed in the *lp*  
 21872 banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be  
 21873 used.

#### 21874 **ASYNCHRONOUS EVENTS**

21875 Default.

#### 21876 **STDOUT**

21877 The *lp* utility shall write a *request ID* to the standard output, unless **-s** is specified. The format of  
 21878 the message is unspecified. The request ID can be used on systems supporting the historical  
 21879 *cancel* and *lpstat* utilities.

#### 21880 **STDERR**

21881 The standard error shall be used only for diagnostic messages.

#### 21882 **OUTPUT FILES**

21883 None.

#### 21884 **EXTENDED DESCRIPTION**

21885 None.

#### 21886 **EXIT STATUS**

21887 The following exit values shall be returned:

- 21888 0 All input files were processed successfully.
- 21889 >0 No output device was available, or an error occurred.

#### 21890 **CONSEQUENCES OF ERRORS**

21891 Default.

#### 21892 **APPLICATION USAGE**

21893 The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's  
 21894 default page size.

21895 A conforming application can use one of the *file* operands only with the **-c** option or if the file is  
 21896 publicly readable and guaranteed to be available at the time of printing. This is because  
 21897 IEEE Std 1003.1-2001 gives the implementation the freedom to queue up the request for printing  
 21898 at some later time by a different process that might not be able to access the file.

#### 21899 **EXAMPLES**

- 21900 1. To print file *file*:  
 21901 `lp -c file`
- 21902 2. To print multiple files with headers:  
 21903 `pr file1 file2 | lp`

#### 21904 **RATIONALE**

21905 The *lp* utility was designed to be a basic version of a utility that is already available in many  
 21906 historical implementations. The standard developers considered that it should be implementable  
 21907 simply as:

21908 `cat "$@" > /dev/lp`

21909 after appropriate processing of options, if that is how the implementation chose to do it and if  
 21910 exclusive access could be granted (so that two users did not write to the device simultaneously).  
 21911 Although in the future the standard developers may add other options to this utility, it should

21912 always be able to execute with no options or operands and send the standard input to an  
 21913 unspecified output device.

21914 This volume of IEEE Std 1003.1-2001 makes no representations concerning the format of the  
 21915 printed output, except that it must be “human-readable” and “non-volatile”. Thus, writing by  
 21916 default to a disk or tape drive or a display terminal would not qualify. (Such destinations are not  
 21917 prohibited when `-d dest`, `LPDEST`, or `PRINTER` are used, however.)

21918 This volume of IEEE Std 1003.1-2001 is worded such that a “print job” consisting of multiple  
 21919 input files, possibly in multiple copies, is guaranteed to print so that any one file is not  
 21920 intermixed with another, but there is no statement that all the files or copies have to print out  
 21921 together.

21922 The `-c` option may imply a spooling operation, but this is not required. The utility can be  
 21923 implemented to wait until the printer is ready and then wait until it is finished. Because of that,  
 21924 there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

21925 On some historical systems, the request ID reported on the STDOUT can be used to later cancel  
 21926 or find the status of a request using utilities not defined in this volume of IEEE Std 1003.1-2001.

21927 Although the historical System V `lp` and BSD `lpr` utilities have provided similar functionality,  
 21928 they used different names for the environment variable specifying the destination printer. Since  
 21929 the name of the utility here is `lp`, `LPDEST` (used by the System V `lp` utility) was given precedence  
 21930 over `PRINTER` (used by the BSD `lpr` utility). Since environments of users frequently contain one  
 21931 or the other environment variable, the `lp` utility is required to recognize both. If this was not  
 21932 done, many applications would send output to unexpected output devices when users moved  
 21933 from system to system.

21934 Some have commented that `lp` has far too little functionality to make it worthwhile. Requests  
 21935 have proposed additional options or operands or both that added functionality. The requests  
 21936 included:

- 21937 • Wording *requiring* the output to be “hardcopy”
- 21938 • A requirement for multiple printers
- 21939 • Options for supporting various page-description languages

21940 Given that a compliant system is not required to even have a printer, placing further restrictions  
 21941 upon the behavior of the printer is not useful. Since hardcopy format is so application-  
 21942 dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that  
 21943 should be required on all compliant systems.

21944 The term *unspecified* is used in this section in lieu of *implementation-defined* as most known  
 21945 implementations would not be able to make definitive statements in their conformance  
 21946 documents; the existence and usage of printers is very dependent on how the system  
 21947 administrator configures each individual system.

21948 Since the default destination, device type, queuing mechanisms, and acceptable forms of input  
 21949 are all unspecified, usage guidelines for what a conforming application can do are as follows:

- 21950 • Use the command in a pipeline, or with `-c`, so that there are no permission problems and the  
 21951 files can be safely deleted or modified.
- 21952 • Limit output to text files of reasonable line lengths and printable characters and include no  
 21953 device-specific formatting information, such as a page description language. The meaning of  
 21954 “reasonable” in this context can only be answered as a quality-of-implementation issue, but  
 21955 it should be apparent from historical usage patterns in the industry and the locale. The `pr` and  
 21956 `fold` utilities can be used to achieve reasonable formatting for the default page size of the

21957 implementation.

21958 Alternatively, the application can arrange its installation in such a way that it requires the  
21959 system administrator or operator to provide the appropriate information on *lp* options and  
21960 environment variable values.

21961 At a minimum, having this utility in this volume of IEEE Std 1003.1-2001 tells the industry that  
21962 conforming applications require a means to print output and provides at least a command name  
21963 and *LPDEST* routing mechanism that can be used for discussions between vendors, application  
21964 writers, and users. The use of “should” in the DESCRIPTION of *lp* clearly shows the intent of  
21965 the standard developers, even if they cannot mandate that all systems (such as laptops) have  
21966 printers.

21967 This volume of IEEE Std 1003.1-2001 does not specify what the ownership of the process  
21968 performing the writing to the output device may be. If *-c* is not used, it is unspecified whether  
21969 the process performing the writing to the output device has permission to read *file* if there are  
21970 any restrictions in place on who may read *file* until after it is printed. Also, if *-c* is not used, the  
21971 results of deleting *file* before it is printed are unspecified.

## 21972 FUTURE DIRECTIONS

21973 None.

## 21974 SEE ALSO

21975 *mailx*

## 21976 CHANGE HISTORY

21977 First released in Issue 2.

## 21978 Issue 6

21979 The following new requirements on POSIX implementations derive from alignment with the  
21980 Single UNIX Specification:

- 21981 • In the DESCRIPTION, the requirement to associate a unique request ID, and the normal  
21982 generation of a banner page is added.
- 21983 • In the OPTIONS section:
  - 21984 — The *-d dest* description is expanded, but references to *lpstat* are removed.
  - 21985 — The *-m*, *-o*, *-s*, *-t*, and *-w* options are added.
- 21986 • In the ENVIRONMENT VARIABLES section, *LC\_TIME* may now affect the execution.
- 21987 • The STDOUT section is added.

21988 The normative text is reworded to avoid use of the term “must” for application requirements.

21989 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

21990 **NAME**

21991       ls — list directory contents

21992 **SYNOPSIS**

21993 xsi       ls [-CFRacdilqrtl] [-H | -L ] [-fgmnopsx] [file...]

21994 **DESCRIPTION**

21995       For each operand that names a file of a type other than directory or symbolic link to a directory,  
 21996       *ls* shall write the name of the file as well as any requested, associated information. For each  
 21997       operand that names a file of type directory, *ls* shall write the names of files contained within the  
 21998       directory as well as any requested, associated information. If one of the **-d**, **-F**, or **-l** options are  
 21999       specified, and one of the **-H** or **-L** options are not specified, for each operand that names a file of  
 22000       type symbolic link to a directory, *ls* shall write the name of the file as well as any requested,  
 22001       associated information. If none of the **-d**, **-F**, or **-l** options are specified, or the **-H** or **-L** options  
 22002       are specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write  
 22003       the names of files contained within the directory as well as any requested, associated  
 22004       information.

22005       If no operands are specified, *ls* shall write the contents of the current directory. If more than one  
 22006       operand is specified, *ls* shall write non-directory operands first; it shall sort directory and non-  
 22007       directory operands separately according to the collating sequence in the current locale.

22008       The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an  
 22009       ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic  
 22010       message to standard error and shall either recover its position in the hierarchy or terminate.

22011 **OPTIONS**

22012       The *ls* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 22013       Utility Syntax Guidelines.

22014       The following options shall be supported:

22015       **-C**       Write multi-text-column output with entries sorted down the columns, according  
 22016       to the collating sequence. The number of text columns and the column separator  
 22017       characters are unspecified, but should be adapted to the nature of the output  
 22018       device.

22019       **-F**       Do not follow symbolic links named as operands unless the **-H** or **-L** options are  
 22020       specified. Write a slash ( '/' ) immediately after each pathname that is a directory,  
 22021       an asterisk ( '\*' ) after each that is executable, a vertical bar ( '|' ) after each that is  
 22022       a FIFO, and an at sign ( '@' ) after each that is a symbolic link. For other file types,  
 22023       other symbols may be written.

22024       **-H**       If a symbolic link referencing a file of type directory is specified on the command  
 22025       line, *ls* shall evaluate the file information and file type to be those of the file  
 22026       referenced by the link, and not the link itself; however, *ls* shall write the name of  
 22027       the link itself and not the file referenced by the link.

22028       **-L**       Evaluate the file information and file type for all symbolic links (whether named  
 22029       on the command line or encountered in a file hierarchy) to be those of the file  
 22030       referenced by the link, and not the link itself; however, *ls* shall write the name of  
 22031       the link itself and not the file referenced by the link. When **-L** is used with **-l**, write  
 22032       the contents of symbolic links in the long format (see the STDOUT section).

22033       **-R**       Recursively list subdirectories encountered.

22034       **-a**       Write out all directory entries, including those whose names begin with a period  
 22035       ( '.' ). Entries beginning with a period shall not be written out unless explicitly

|           |           |                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 22036     |           | referenced, the <b>-a</b> option is supplied, or an implementation-defined condition shall cause them to be written.                                                                                                                                                                                                                                                          |
| 22037     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22038     | <b>-c</b> | Use time of last modification of the file status information (see < <b>sys/stat.h</b> > in the System Interfaces volume of IEEE Std 1003.1-2001) instead of last modification of the file itself for sorting ( <b>-t</b> ) or writing ( <b>-l</b> ).                                                                                                                          |
| 22039     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22040     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22041     | <b>-d</b> | Do not follow symbolic links named as operands unless the <b>-H</b> or <b>-L</b> options are specified. Do not treat directories differently than other types of files. The use of <b>-d</b> with <b>-R</b> produces unspecified results.                                                                                                                                     |
| 22042     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22043     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22044 XSI | <b>-f</b> | Force each argument to be interpreted as a directory and list the name found in each slot. This option shall turn off <b>-l</b> , <b>-t</b> , <b>-s</b> , and <b>-r</b> , and shall turn on <b>-a</b> ; the order is the order in which entries appear in the directory.                                                                                                      |
| 22045     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22046     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22047 XSI | <b>-g</b> | The same as <b>-l</b> , except that the owner shall not be written.                                                                                                                                                                                                                                                                                                           |
| 22048     | <b>-i</b> | For each file, write the file's file serial number (see <i>stat()</i> in the System Interfaces volume of IEEE Std 1003.1-2001).                                                                                                                                                                                                                                               |
| 22049     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22050     | <b>-l</b> | (The letter ell.) Do not follow symbolic links named as operands unless the <b>-H</b> or <b>-L</b> options are specified. Write out in long format (see the STDOUT section). When <b>-l</b> (ell) is specified, <b>-1</b> (one) shall be assumed.                                                                                                                             |
| 22051     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22052     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22053 XSI | <b>-m</b> | Stream output format; list files across the page, separated by commas.                                                                                                                                                                                                                                                                                                        |
| 22054 XSI | <b>-n</b> | The same as <b>-l</b> , except that the owner's UID and GID numbers shall be written, rather than the associated character strings.                                                                                                                                                                                                                                           |
| 22055     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22056 XSI | <b>-o</b> | The same as <b>-l</b> , except that the group shall not be written.                                                                                                                                                                                                                                                                                                           |
| 22057 XSI | <b>-p</b> | Write a slash ( <b>/</b> ) after each filename if that file is a directory.                                                                                                                                                                                                                                                                                                   |
| 22058     | <b>-q</b> | Force each instance of non-printable filename characters and <tab>s to be written as the question-mark ( <b>?</b> ) character. Implementations may provide this option by default if the output is to a terminal device.                                                                                                                                                      |
| 22059     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22060     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22061     | <b>-r</b> | Reverse the order of the sort to get reverse collating sequence or oldest first.                                                                                                                                                                                                                                                                                              |
| 22062 XSI | <b>-s</b> | Indicate the total number of file system blocks consumed by each file displayed. The block size is implementation-defined.                                                                                                                                                                                                                                                    |
| 22063     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22064     | <b>-t</b> | Sort with the primary key being time modified (most recently modified first) and the secondary key being filename in the collating sequence.                                                                                                                                                                                                                                  |
| 22065     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22066     | <b>-u</b> | Use time of last access (see < <b>sys/stat.h</b> >) instead of last modification of the file for sorting ( <b>-t</b> ) or writing ( <b>-l</b> ).                                                                                                                                                                                                                              |
| 22067     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22068 XSI | <b>-x</b> | The same as <b>-C</b> , except that the multi-text-column output is produced with entries sorted across, rather than down, the columns.                                                                                                                                                                                                                                       |
| 22069     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22070     | <b>-1</b> | (The numeric digit one.) Force output to be one entry per line.                                                                                                                                                                                                                                                                                                               |
| 22071     |           | Specifying more than one of the options in the following mutually-exclusive pairs shall not be considered an error: <b>-C</b> and <b>-l</b> (ell), <b>-m</b> and <b>-l</b> (ell), <b>-x</b> and <b>-l</b> (ell), <b>-C</b> and <b>-1</b> (one), <b>-H</b> and <b>-L</b> , <b>-c</b> and <b>-u</b> . The last option specified in each pair shall determine the output format. |
| 22072 XSI |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 22073     |           |                                                                                                                                                                                                                                                                                                                                                                               |

## 22074 OPERANDS

22075 The following operand shall be supported:

|       |             |                                                                                                                                 |
|-------|-------------|---------------------------------------------------------------------------------------------------------------------------------|
| 22076 | <i>file</i> | A pathname of a file to be written. If the file specified is not found, a diagnostic message shall be output on standard error. |
| 22077 |             |                                                                                                                                 |



22078 **STDIN**

22079 Not used.

22080 **INPUT FILES**

22081 None.

22082 **ENVIRONMENT VARIABLES**22083 The following environment variables shall affect the execution of *ls*:

22084 **COLUMNS** Determine the user's preferred column position width for writing multiple text-column output. If this variable contains a string representing a decimal integer, the  
 22085 *ls* utility shall calculate how many pathname text columns to write (see **-C**) based  
 22086 on the width provided. If **COLUMNS** is not set or invalid, an implementation-  
 22087 defined number of column positions shall be assumed, based on the  
 22088 implementation's knowledge of the output device. The column width chosen to  
 22089 write the names of files in any given directory shall be constant. Filenames shall  
 22090 not be truncated to fit into the multiple text-column output.  
 22091

22092 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 22093 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 22094 Internationalization Variables for the precedence of internationalization variables  
 22095 used to determine the values of locale categories.)

22096 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 22097 internationalization variables.

22098 **LC\_COLLATE**  
 22099 Determine the locale for character collation information in determining the  
 22100 pathname collation sequence.

22101 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 22102 characters (for example, single-byte as opposed to multi-byte characters in  
 22103 arguments) and which characters are defined as printable (character class **print**).

22104 **LC\_MESSAGES**  
 22105 Determine the locale that should be used to affect the format and contents of  
 22106 diagnostic messages written to standard error.

22107 **LC\_TIME** Determine the format and contents for date and time strings written by *ls*.

22108 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

22109 **TZ** Determine the timezone for date and time strings written by *ls*. If **TZ** is unset or  
 22110 null, an unspecified default timezone shall be used.

22111 **ASYNCHRONOUS EVENTS**

22112 Default.

22113 **STDOUT**

22114 The default format shall be to list one entry per line to standard output; the exceptions are to  
 22115 XSI terminals or when one of the **-C**, **-m**, or **-x** options is specified. If the output is to a terminal, the  
 22116 format is implementation-defined.

22117 XSI When **-m** is specified, the format used shall be:

22118 "%s, %s, ... \n", <filename1>, <filename2>

22119 where the largest number of filenames shall be written without exceeding the length of the line.

22120 If the **-i** option is specified, the file's file serial number (see <sys/stat.h>) shall be written in the  
 22121 following format before any other output for the corresponding entry:

22122        %u ", <file serial number>

22123        If the **-l** option is specified without **-L**, the following information shall be written:

22124        "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,  
 22125                <owner name>, <group name>, <number of bytes in the file>,  
 22126                <date and time>, <pathname>

22127        If the file is a symbolic link, this information shall be about the link itself and the <pathname>  
 22128        field shall be of the form:

22129        "%s -> %s", <pathname of link>, <contents of link>

22130        If both **-l** and **-L** are specified, the following information shall be written:

22131        "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,  
 22132                <owner name>, <group name>, <number of bytes in the file>,  
 22133                <date and time>, <pathname of link>

22134        where all fields except <pathname of link> shall be for the file resolved from the symbolic link.

22135 xSI        The **-g**, **-n**, and **-o** options use the same format as **-l**, but with omitted items and their  
 22136        associated <blank>s. See the OPTIONS section.

22137 xSI        In both the preceding **-l** forms, if <owner name> or <group name> cannot be determined, or if **-n**  
 22138        is given, they shall be replaced with their associated numeric values using the format %u.

22139        The <date and time> field shall contain the appropriate date and timestamp of when the file was  
 22140        last modified. In the POSIX locale, the field shall be the equivalent of the output of the following  
 22141        *date* command:

22142        date "+%b %e %H:%M"

22143        if the file has been modified in the last six months, or:

22144        date "+%b %e %Y"

22145        (where two <space>s are used between %e and %Y) if the file has not been modified in the last six  
 22146        months or if the modification date is in the future, except that, in both cases, the final <newline>  
 22147        produced by *date* shall not be included and the output shall be as if the *date* command were  
 22148        executed at the time of the last modification date of the file rather than the current time. When  
 22149        the *LC\_TIME* locale category is not set to the POSIX locale, a different format and order of  
 22150        presentation of this field may be used.

22151        If the file is a character special or block special file, the size of the file may be replaced with  
 22152        implementation-defined information associated with the device in question.

22153        If the pathname was specified as a *file* operand, it shall be written as specified.

22154 xSI        The file mode written under the **-l**, **-g**, **-n**, and **-o** options shall consist of the following format:

22155        "%c%s%s%s%c", <entry type>, <owner permissions>,  
 22156                <group permissions>, <other permissions>,  
 22157                <optional alternate access method flag>

22158        The <optional alternate access method flag> shall be a single <space> if there is no alternate or  
 22159        additional access control method associated with the file; otherwise, a printable character shall  
 22160        be used.

22161        The <entry type> character shall describe the type of file, as follows:

22162        d        Directory.

|           |                                                                                                                                                   |                                                                                            |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| 22163     | b                                                                                                                                                 | Block special file.                                                                        |
| 22164     | c                                                                                                                                                 | Character special file.                                                                    |
| 22165     | l (ell)                                                                                                                                           | Symbolic link.                                                                             |
| 22166     | p                                                                                                                                                 | FIFO.                                                                                      |
| 22167     | –                                                                                                                                                 | Regular file.                                                                              |
| 22168     | Implementations may add other characters to this list to represent other implementation-defined                                                   |                                                                                            |
| 22169     | file types.                                                                                                                                       |                                                                                            |
| 22170     | The next three fields shall be three characters each:                                                                                             |                                                                                            |
| 22171     | <owner permissions>                                                                                                                               |                                                                                            |
| 22172     | Permissions for the file owner class (see the Base Definitions volume of                                                                          |                                                                                            |
| 22173     | IEEE Std 1003.1-2001, Section 4.4, File Access Permissions).                                                                                      |                                                                                            |
| 22174     | <group permissions>                                                                                                                               |                                                                                            |
| 22175     | Permissions for the file group class.                                                                                                             |                                                                                            |
| 22176     | <other permissions>                                                                                                                               |                                                                                            |
| 22177     | Permissions for the file other class.                                                                                                             |                                                                                            |
| 22178     | Each field shall have three character positions:                                                                                                  |                                                                                            |
| 22179     | 1.                                                                                                                                                | If 'r', the file is readable; if '–', the file is not readable.                            |
| 22180     | 2.                                                                                                                                                | If 'w', the file is writable; if '–', the file is not writable.                            |
| 22181     | 3.                                                                                                                                                | The first of the following that applies:                                                   |
| 22182     | S                                                                                                                                                 | If in <owner permissions>, the file is not executable and set-user-ID mode is set. If in   |
| 22183     |                                                                                                                                                   | <group permissions>, the file is not executable and set-group-ID mode is set.              |
| 22184     | s                                                                                                                                                 | If in <owner permissions>, the file is executable and set-user-ID mode is set. If in       |
| 22185     |                                                                                                                                                   | <group permissions>, the file is executable and set-group-ID mode is set.                  |
| 22186 XSI | T                                                                                                                                                 | If in <other permissions> and the file is a directory, search permission is not granted to |
| 22187     |                                                                                                                                                   | others, and the restricted deletion flag is set.                                           |
| 22188 XSI | t                                                                                                                                                 | If in <other permissions> and the file is a directory, search permission is granted to     |
| 22189     |                                                                                                                                                   | others, and the restricted deletion flag is set.                                           |
| 22190     | x                                                                                                                                                 | The file is executable or the directory is searchable.                                     |
| 22191     | –                                                                                                                                                 | None of the attributes of 'S', 's', 'T', 't', or 'x' applies.                              |
| 22192     | Implementations may add other characters to this list for the third character position. Such                                                      |                                                                                            |
| 22193     | additions shall, however, be written in lowercase if the file is executable or searchable, and                                                    |                                                                                            |
| 22194     | in uppercase if it is not.                                                                                                                        |                                                                                            |
| 22195 XSI | If any of the <b>–l</b> , <b>–g</b> , <b>–n</b> , <b>–o</b> , or <b>–s</b> options is specified, each list of files within the directory shall be |                                                                                            |
| 22196     | preceded by a status line indicating the number of file system blocks occupied by files in the                                                    |                                                                                            |
| 22197     | directory in 512-byte units, rounded up to the next integral number of units, if necessary. In the                                                |                                                                                            |
| 22198     | POSIX locale, the format shall be:                                                                                                                |                                                                                            |
| 22199     | <code>"total %u\n", &lt;number of units in the directory&gt;</code>                                                                               |                                                                                            |
| 22200     | If more than one directory, or a combination of non-directory files and directories are written,                                                  |                                                                                            |
| 22201     | either as a result of specifying multiple operands, or the <b>–R</b> option, each list of files within a                                          |                                                                                            |
| 22202     | directory shall be preceded by:                                                                                                                   |                                                                                            |

22203        "`\n%s:\n`", *<directory name>*

22204        If this string is the first thing to be written, the first *<newline>* shall not be written. This output  
22205        shall precede the number of units in the directory.

22206 xSI      If the `-s` option is given, each file shall be written with the number of blocks used by the file.  
22207        Along with `-C`, `-l`, `-m`, or `-x`, the number and a *<space>* shall precede the filename; with `-g`, `-l`,  
22208        `-n`, or `-o`, they shall precede each line describing a file.

## 22209 **STDERR**

22210        The standard error shall be used only for diagnostic messages.

## 22211 **OUTPUT FILES**

22212        None.

## 22213 **EXTENDED DESCRIPTION**

22214        None.

## 22215 **EXIT STATUS**

22216        The following exit values shall be returned:

22217            0   Successful completion.

22218            >0   An error occurred.

## 22219 **CONSEQUENCES OF ERRORS**

22220        Default.

## 22221 **APPLICATION USAGE**

22222        Many implementations use the equal sign (`'='`) to denote sockets bound to the file system for  
22223        the `-F` option. Similarly, many historical implementations use the `'s'` character to denote  
22224        sockets as the entry type characters for the `-l` option.

22225        It is difficult for an application to use every part of the file modes field of `ls -l` in a portable  
22226        manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as  
22227        implementations may have extensions. Applications can use this field to pass directly to a user  
22228        printout or prompt, but actions based on its contents should generally be deferred, instead, to  
22229        the *test* utility.

22230        The output of *ls* (with the `-l` and related options) contains information that logically could be  
22231        used by utilities such as *chmod* and *touch* to restore files to a known state. However, this  
22232        information is presented in a format that cannot be used directly by those utilities or be easily  
22233        translated into a format that can be used. A character has been added to the end of the  
22234        permissions string so that applications at least have an indication that they may be working in  
22235        an area they do not understand instead of assuming that they can translate the permissions  
22236        string into something that can be used. Future issues or related documents may define one or  
22237        more specific characters to be used based on different standard additional or alternative access  
22238        control mechanisms.

22239        As with many of the utilities that deal with filenames, the output of *ls* for multiple files or in one  
22240        of the long listing formats must be used carefully on systems where filenames can contain  
22241        embedded white space. Systems and system administrators should institute policies and user  
22242        training to limit the use of such filenames.

22243        The number of disk blocks occupied by the file that it reports varies depending on underlying  
22244        file system type, block size units reported, and the method of calculating the number of blocks.  
22245        On some file system types, the number is the actual number of blocks occupied by the file  
22246        (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the  
22247        file size (usually making an allowance for indirect blocks, but ignoring holes).

22248 **EXAMPLES**

22249 An example of a small directory tree being fully listed with *ls -laRF a* in the POSIX locale:

```

22250 total 11
22251 drwxr-xr-x 3 hlj prog 64 Jul 4 12:07 ./
22252 drwxrwxrwx 4 hlj prog 3264 Jul 4 12:09 ../
22253 drwxr-xr-x 2 hlj prog 48 Jul 4 12:07 b/
22254 -rwxr--r-- 1 hlj prog 572 Jul 4 12:07 foo*

22255 a/b:
22256 total 4
22257 drwxr-xr-x 2 hlj prog 48 Jul 4 12:07 ./
22258 drwxr-xr-x 3 hlj prog 64 Jul 4 12:07 ../
22259 -rw-r--r-- 1 hlj prog 700 Jul 4 12:07 bar

```

22260 **RATIONALE**

22261 Some historical implementations of the *ls* utility show all entries in a directory except dot and  
 22262 dot-dot when a superuser invokes *ls* without specifying the **-a** option. When “normal” users  
 22263 invoke *ls* without specifying **-a**, they should not see information about any files with names  
 22264 beginning with a period unless they were named as *file* operands.

22265 Implementations are expected to traverse arbitrary depths when processing the **-R** option. The  
 22266 only limitation on depth should be based on running out of physical storage for keeping track of  
 22267 untraversed directories.

22268 The **-1** (one) option was historically found in BSD and BSD-derived implementations only. It is  
 22269 required in this volume of IEEE Std 1003.1-2001 so that conforming applications might ensure  
 22270 that output is one entry per line, even if the output is to a terminal.

22271 Generally, this volume of IEEE Std 1003.1-2001 is silent about what happens when options are  
 22272 given multiple times. In the cases of **-C**, **-l**, and **-1**, however, it does specify the results of these  
 22273 overlapping options. Since *ls* is one of the most aliased commands, it is important that the  
 22274 implementation perform intuitively. For example, if the alias were:

```
22275 alias ls="ls -C"
```

22276 and the user typed *ls -1*, single-text-column output should result, not an error.

22277 The BSD *ls* provides a **-A** option (like **-a**, but dot and dot-dot are not written out). The small  
 22278 difference from **-a** did not seem important enough to require both.

22279 Implementations may make **-q** the default for terminals to prevent trojan horse attacks on  
 22280 terminals with special escape sequences. This is not required because:

- 22281 • Some control characters may be useful on some terminals; for example, a system might write  
 22282 them as "\001" or "^A".
- 22283 • Special behavior for terminals is not relevant to applications portability.

22284 An early proposal specified that the optional alternate access method flag had to be '+' if there  
 22285 was an alternate access method used on the file or <space> if there was not. This was changed to  
 22286 be <space> if there is not and a single printable character if there is. This was done for three  
 22287 reasons:

- 22288 1. There are historical implementations using characters other than '+'.
- 22289 2. There are implementations that vary this character used in that position to distinguish  
 22290 between various alternate access methods in use.

- 22291 3. The standard developers did not want to preclude future specifications that might need a  
22292 way to specify more than one alternate access method.
- 22293 Nonetheless, implementations providing a single alternate access method are encouraged to use  
22294 ' + '.
- 22295 In an early proposal, the units used to specify the number of blocks occupied by files in a  
22296 directory in an *ls -l* listing were implementation-defined. This was because BSD systems have  
22297 historically used 1024-byte units and System V systems have historically used 512-byte units. It  
22298 was pointed out by BSD developers that their system has used 512-byte units in some places and  
22299 1024-byte units in other places. (System V has consistently used 512.) Therefore, this volume of  
22300 IEEE Std 1003.1-2001 usually specifies 512. Future releases of BSD are expected to consistently  
22301 provide 512 bytes as a default with a way of specifying 1024-byte units where appropriate.
- 22302 The *<date and time>* field in the *-l* format is specified only for the POSIX locale. As noted, the  
22303 format can be different in other locales. No mechanism for defining this is present in this volume  
22304 of IEEE Std 1003.1-2001, as the appropriate vehicle is a messaging system; that is, the format  
22305 should be specified as a “message”.
- 22306 **FUTURE DIRECTIONS**
- 22307 The *-s* uses implementation-defined units and cannot be used portably; it may be withdrawn in  
22308 a future version.
- 22309 **SEE ALSO**
- 22310 *chmod*, *find*, the System Interfaces volume of IEEE Std 1003.1-2001, *stat()*, the Base Definitions  
22311 volume of IEEE Std 1003.1-2001, **<sys/stat.h>**
- 22312 **CHANGE HISTORY**
- 22313 First released in Issue 2.
- 22314 **Issue 5**
- 22315 A second FUTURE DIRECTION is added.
- 22316 **Issue 6**
- 22317 The following new requirements on POSIX implementations derive from alignment with the  
22318 Single UNIX Specification:
- 22319 • In the *-F* option, other symbols are allowed for other file types.
- 22320 Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.
- 22321 The Open Group Base Resolution bwg2001-010 is applied, adding the *T* and *t* fields as an XSI  
22322 extension.

22323 **NAME**22324           m4 — macro processor (**DEVELOPMENT**)22325 **SYNOPSIS**22326 xSI        m4 [-s] [-D *name*[=*val*]] ... [-U *name*] ... *file*...

22327

22328 **DESCRIPTION**

22329       The *m4* utility is a macro processor that shall read one or more text files, process them according  
 22330       to their included macro statements, and write the results to standard output.

22331 **OPTIONS**

22332       The *m4* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 22333       Utility Syntax Guidelines, except that the order of the **-D** and **-U** options shall be significant.

22334       The following options shall be supported:

22335       **-s**           Enable line synchronization output for the c99 preprocessor phase (that is, **#line**  
 22336       directives).

22337       **-D *name*[=*val*]**   Define *name* to *val* or to null if *=val* is omitted.  
 22338

22339       **-U *name***       Undefine *name*.

22340 **OPERANDS**

22341       The following operand shall be supported:

22342       *file*           A pathname of a text file to be processed. If no *file* is given, or if it is '-', the  
 22343       standard input shall be read.

22344 **STDIN**

22345       The standard input shall be a text file that is used if no *file* operand is given, or if it is '-'.

22346 **INPUT FILES**

22347       The input file named by the *file* operand shall be a text file.

22348 **ENVIRONMENT VARIABLES**

22349       The following environment variables shall affect the execution of *m4*:

22350       **LANG**           Provide a default value for the internationalization variables that are unset or null.  
 22351       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 22352       Internationalization Variables for the precedence of internationalization variables  
 22353       used to determine the values of locale categories.)

22354       **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 22355       internationalization variables.

22356       **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
 22357       characters (for example, single-byte as opposed to multi-byte characters in  
 22358       arguments and input files).

22359       **LC\_MESSAGES**   Determine the locale that should be used to affect the format and contents of  
 22360       diagnostic messages written to standard error.  
 22361

22362       **NLSPATH**   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

22363 **ASYNCHRONOUS EVENTS**

22364 Default.

22365 **STDOUT**

22366 The standard output shall be the same as the input files, after being processed for macro  
 22367 expansion.

22368 **STDERR**

22369 The standard error shall be used to display strings with the **errprint** macro, macro tracing  
 22370 enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for  
 22371 diagnostic messages.

22372 **OUTPUT FILES**

22373 None.

22374 **EXTENDED DESCRIPTION**

22375 The *m4* utility shall compare each token from the input against the set of built-in and user-  
 22376 defined macros. If the token matches the name of a macro, then the token shall be replaced by  
 22377 the macro's defining text, if any, and rescanned for matching macro names. Once no portion of  
 22378 the token matches the name of a macro, it shall be written to standard output. Macros may have  
 22379 arguments, in which case the arguments shall be substituted into the defining text before it is  
 22380 rescanned.

22381 Macro calls have the form:

22382 *name(arg1, arg2, ..., argn)*

22383 Macro names shall consist of letters, digits, and underscores, where the first character is not a  
 22384 digit. Tokens not of this form shall not be treated as macros.

22385 The application shall ensure that the left parenthesis immediately follows the name of the  
 22386 macro. If a token matching the name of a macro is not followed by a left parenthesis, it is  
 22387 handled as a use of that macro without arguments.

22388 If a macro name is followed by a left parenthesis, its arguments are the comma-separated tokens  
 22389 between the left parenthesis and the matching right parenthesis. Unquoted <blank>s and  
 22390 <newline>s preceding each argument shall be ignored. All other characters, including trailing  
 22391 <blank>s and <newline>s, are retained. Commas enclosed between left and right parenthesis  
 22392 characters do not delimit arguments.

22393 Arguments are positionally defined and referenced. The string "\$1" in the defining text shall be  
 22394 replaced by the first argument. Systems shall support at least nine arguments; only the first nine  
 22395 can be referenced, using the strings "\$1" to "\$9", inclusive. The string "\$0" is replaced with  
 22396 the name of the macro. The string "\$#" is replaced by the number of arguments as a string. The  
 22397 string "\$\*" is replaced by a list of all of the arguments, separated by commas. The string "\$@"  
 22398 is replaced by a list of all of the arguments separated by commas, and each argument is quoted  
 22399 using the current left and right quoting strings.

22400 If fewer arguments are supplied than are in the macro definition, the omitted arguments are  
 22401 taken to be null. It is not an error if more arguments are supplied than are in the macro  
 22402 definition.

22403 No special meaning is given to any characters enclosed between matching left and right quoting  
 22404 strings, but the quoting strings are themselves discarded. By default, the left quoting string  
 22405 consists of a grave accent ('`') and the right quoting string consists of an acute accent ('`');  
 22406 see also the **changequote** macro.

22407 Comments are written but not scanned for matching macro names; by default, the begin-  
 22408 comment string consists of the number sign character and the end-comment string consists of a



|       |                    |                                                                                                                                              |
|-------|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| 22409 |                    | <newline>. See also the <b>changecom</b> and <b>dnl</b> macros.                                                                              |
| 22410 |                    | The <i>m4</i> utility shall make available the following built-in macros. They can be redefined, but                                         |
| 22411 |                    | once this is done the original meaning is lost. Their values shall be null unless otherwise stated.                                          |
| 22412 |                    | In the descriptions below, the term <i>defining text</i> refers to the value of the macro: the second                                        |
| 22413 |                    | argument to the <b>define</b> macro, among other things. Except for the first argument to the <b>eval</b>                                    |
| 22414 |                    | macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The                                                  |
| 22415 |                    | string values produced as the defining text of the <b>decr</b> , <b>divnum</b> , <b>incr</b> , <b>index</b> , <b>len</b> , and <b>sysval</b> |
| 22416 |                    | built-in macros shall be in the form of a decimal-constant as defined in the C language.                                                     |
| 22417 | <b>changecom</b>   | The <b>changecom</b> macro shall set the begin-comment and end-comment strings.                                                              |
| 22418 |                    | With no arguments, the comment mechanism shall be disabled. With a single                                                                    |
| 22419 |                    | argument, that argument shall become the begin-comment string and the                                                                        |
| 22420 |                    | <newline> shall become the end-comment string. With two arguments, the first                                                                 |
| 22421 |                    | argument shall become the begin-comment string and the second argument shall                                                                 |
| 22422 |                    | become the end-comment string. Systems shall support comment strings of at least                                                             |
| 22423 |                    | five characters.                                                                                                                             |
| 22424 | <b>changequote</b> | The <b>changequote</b> macro shall set the begin-quote and end-quote strings. With no                                                        |
| 22425 |                    | arguments, the quote strings shall be set to the default values (that is, ' '). With a                                                       |
| 22426 |                    | single argument, that argument shall become the begin-quote string and the                                                                   |
| 22427 |                    | <newline> shall become the end-quote string. With two arguments, the first                                                                   |
| 22428 |                    | argument shall become the begin-quote string and the second argument shall                                                                   |
| 22429 |                    | become the end-quote string. Systems shall support quote strings of at least five                                                            |
| 22430 |                    | characters.                                                                                                                                  |
| 22431 | <b>decr</b>        | The defining text of the <b>decr</b> macro shall be its first argument decremented by 1. It                                                  |
| 22432 |                    | shall be an error to specify an argument containing any non-numeric characters.                                                              |
| 22433 | <b>define</b>      | The second argument shall become the defining text of the macro whose name is                                                                |
| 22434 |                    | the first argument.                                                                                                                          |
| 22435 | <b>defn</b>        | The defining text of the <b>defn</b> macro shall be the quoted definition (using the                                                         |
| 22436 |                    | current quoting strings) of its arguments.                                                                                                   |
| 22437 | <b>divert</b>      | The <i>m4</i> utility maintains nine temporary buffers, numbered 1 to 9, inclusive. When                                                     |
| 22438 |                    | the last of the input has been processed, any output that has been placed in these                                                           |
| 22439 |                    | buffers shall be written to standard output in buffer-numerical order. The <b>divert</b>                                                     |
| 22440 |                    | macro shall divert future output to the buffer specified by its argument. Specifying                                                         |
| 22441 |                    | no argument or an argument of 0 shall resume the normal output process. Output                                                               |
| 22442 |                    | diverted to a stream other than 0 to 9 shall be discarded. It shall be an error to                                                           |
| 22443 |                    | specify an argument containing any non-numeric characters.                                                                                   |
| 22444 | <b>divnum</b>      | The defining text of the <b>divnum</b> macro shall be the number of the current output                                                       |
| 22445 |                    | stream as a string.                                                                                                                          |
| 22446 | <b>dnl</b>         | The <b>dnl</b> macro shall cause <i>m4</i> to discard all input characters up to and including                                               |
| 22447 |                    | the next <newline>.                                                                                                                          |
| 22448 | <b>dumpdef</b>     | The <b>dumpdef</b> macro shall write the defined text to standard error for each of the                                                      |
| 22449 |                    | macros specified as arguments, or, if no arguments are specified, for all macros.                                                            |
| 22450 | <b>errprint</b>    | The <b>errprint</b> macro shall write its arguments to standard error.                                                                       |
| 22451 | <b>eval</b>        | The <b>eval</b> macro shall evaluate its first argument as an arithmetic expression, using                                                   |
| 22452 |                    | 32-bit signed integer arithmetic. All of the C-language operators shall be                                                                   |
| 22453 |                    | supported, except for:                                                                                                                       |

|       |                 |                                                                                               |
|-------|-----------------|-----------------------------------------------------------------------------------------------|
| 22454 |                 | [                                                                                             |
| 22455 |                 | →                                                                                             |
| 22456 |                 | ++                                                                                            |
| 22457 |                 | --                                                                                            |
| 22458 |                 | ( <i>type</i> )                                                                               |
| 22459 |                 | unary *                                                                                       |
| 22460 |                 | sizeof                                                                                        |
| 22461 |                 | ,                                                                                             |
| 22462 |                 | .                                                                                             |
| 22463 |                 | ?:                                                                                            |
| 22464 |                 | unary &                                                                                       |
| 22465 |                 | and all assignment operators. It shall be an error to specify any of these operators.         |
| 22466 |                 | Precedence and associativity shall be as in the ISO C standard. Systems shall                 |
| 22467 |                 | support octal and hexadecimal numbers as in the ISO C standard. The second                    |
| 22468 |                 | argument, if specified, shall set the radix for the result; the default is 10. The third      |
| 22469 |                 | argument, if specified, sets the minimum number of digits in the result. It shall be          |
| 22470 |                 | an error to specify the second or third argument containing any non-numeric                   |
| 22471 |                 | characters.                                                                                   |
| 22472 | <b>ifdef</b>    | If the first argument to the <b>ifdef</b> macro is defined, the defining text shall be the    |
| 22473 |                 | second argument. Otherwise, the defining text shall be the third argument, if                 |
| 22474 |                 | specified, or the null string, if not.                                                        |
| 22475 | <b>ifelse</b>   | The <b>ifelse</b> macro takes three or more arguments. If the first two arguments             |
| 22476 |                 | compare as equal strings (after macro expansion of both arguments), the defining              |
| 22477 |                 | text shall be the third argument. If the first two arguments do not compare as                |
| 22478 |                 | equal strings and there are three arguments, the defining text shall be null. If the          |
| 22479 |                 | first two arguments do not compare as equal strings and there are four or five                |
| 22480 |                 | arguments, the defining text shall be the fourth argument. If the first two                   |
| 22481 |                 | arguments do not compare as equal strings and there are six or more arguments,                |
| 22482 |                 | the first three arguments shall be discarded and processing shall restart with the            |
| 22483 |                 | remaining arguments.                                                                          |
| 22484 | <b>include</b>  | The defining text for the <b>include</b> macro shall be the contents of the file named by     |
| 22485 |                 | the first argument. It shall be an error if the file cannot be read.                          |
| 22486 | <b>incr</b>     | The defining text of the <b>incr</b> macro shall be its first argument incremented by 1. It   |
| 22487 |                 | shall be an error to specify an argument containing any non-numeric characters.               |
| 22488 | <b>index</b>    | The defining text of the <b>index</b> macro shall be the first character position (as a       |
| 22489 |                 | string) in the first argument where a string matching the second argument begins              |
| 22490 |                 | (zero origin), or -1 if the second argument does not occur.                                   |
| 22491 | <b>len</b>      | The defining text of the <b>len</b> macro shall be the length (as a string) of the first      |
| 22492 |                 | argument.                                                                                     |
| 22493 | <b>m4exit</b>   | Exit from the <i>m4</i> utility. If the first argument is specified, it is the exit code. The |
| 22494 |                 | default is zero. It shall be an error to specify an argument containing any non-              |
| 22495 |                 | numeric characters.                                                                           |
| 22496 | <b>m4wrap</b>   | The first argument shall be processed when EOF is reached. If the <b>m4wrap</b> macro         |
| 22497 |                 | is used multiple times, the arguments specified shall be processed in the order in            |
| 22498 |                 | which the <b>m4wrap</b> macros were processed.                                                |
| 22499 | <b>maketemp</b> | The defining text shall be the first argument, with any trailing 'X' characters               |
| 22500 |                 | replaced with the current process ID as a string.                                             |

|       |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 22501 | <b>popdef</b>      | The <b>popdef</b> macro shall delete the current definition of its arguments, replacing that definition with the previous one. If there is no previous definition, the macro is undefined.                                                                                                                                                                                                                                                                                                                                                                                                              |
| 22502 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22503 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22504 | <b>pushdef</b>     | The <b>pushdef</b> macro shall be equivalent to the <b>define</b> macro with the exception that it shall preserve any current definition for future retrieval using the <b>popdef</b> macro.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 22505 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22506 | <b>shift</b>       | The defining text for the <b>shift</b> macro shall be all of its arguments except for the first one.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 22507 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22508 | <b>sinclude</b>    | The <b>sinclude</b> macro shall be equivalent to the <b>include</b> macro, except that it shall not be an error if the file is inaccessible.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 22509 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22510 | <b>substr</b>      | The defining text for the <b>substr</b> macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters. |
| 22511 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22512 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22513 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22514 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22515 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22516 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22517 | <b>syscmd</b>      | The <b>syscmd</b> macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. No output redirection shall be performed by the <i>m4</i> utility. The exit status value from the command can be retrieved using the <b>sysval</b> macro.                                                                                                                                                                                                                                                                                             |
| 22518 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22519 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22520 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22521 | <b>sysval</b>      | The defining text of the <b>sysval</b> macro shall be the exit value of the utility last invoked by the <b>syscmd</b> macro (as a string).                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 22522 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22523 | <b>traceon</b>     | The <b>traceon</b> macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.                                                                                                                                                                                                                                                                                                                                                                                   |
| 22524 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22525 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22526 | <b>traceoff</b>    | The <b>traceoff</b> macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 22527 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22528 | <b>translit</b>    | The defining text of the <b>translit</b> macro shall be the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument.                                                                                                                                                                                                                                                                                                                                                                                                   |
| 22529 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22530 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22531 | <b>undefine</b>    | The <b>undefine</b> macro shall delete all definitions (including those preserved using the <b>pushdef</b> macro) of the macros named by its arguments.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 22532 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22533 | <b>undivert</b>    | The <b>undivert</b> macro shall cause immediate output of any text in temporary buffers named as arguments, or all temporary buffers if no arguments are specified. Buffers can be undiverted into other temporary buffers. Undiverting shall discard the contents of the temporary buffer. It shall be an error to specify an argument containing any non-numeric characters.                                                                                                                                                                                                                          |
| 22534 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22535 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22536 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22537 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22538 | <b>EXIT STATUS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22539 |                    | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 22540 | 0                  | Successful completion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 22541 | >0                 | An error occurred                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 22542 |                    | If the <b>m4exit</b> macro is used, the exit value can be specified by the input file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

22543 **CONSEQUENCES OF ERRORS**

22544 Default.

22545 **APPLICATION USAGE**22546 The **defn** macro is useful for renaming macros, especially built-ins.22547 **EXAMPLES**22548 If the file **m4src** contains the lines:

```

22549 The value of 'VER' is "VER".
22550 ifdef('VER', ``VER'' is defined to be VER., VER is not defined.)
22551 ifelse(VER, 1, ``VER'' is 'VER'.)
22552 ifelse(VER, 2, ``VER'' is 'VER'., ``VER'' is not 2.)
22553 end

```

22554 then the command

22555 `m4 m4src`

22556 or the command:

22557 `m4 -U VER m4src`

22558 produces the output:

```

22559 The value of VER is "VER".
22560 VER is not defined.

22561 VER is not 2.
22562 end

```

22563 The command:

22564 `m4 -D VER m4src`

22565 produces the output:

```

22566 The value of VER is ".
22567 VER is defined to be .

22568 VER is not 2.
22569 end

```

22570 The command:

22571 `m4 -D VER=1 m4src`

22572 produces the output:

```

22573 The value of VER is "1".
22574 VER is defined to be 1.
22575 VER is 1.
22576 VER is not 2.
22577 end

```

22578 The command:

22579 `m4 -D VER=2 m4src`

22580 produces the output:

```

22581 The value of VER is "2".
22582 VER is defined to be 2.

```

22583 VER is 2.  
22584 end

22585 **RATIONALE**

22586 None.

22587 **FUTURE DIRECTIONS**

22588 None.

22589 **SEE ALSO**

22590 c99

22591 **CHANGE HISTORY**

22592 First released in Issue 2.

22593 **Issue 5**

22594 The phrase “the defined text for macros written by the **dumpdef** macro” is added to the  
22595 description of STDERR, and the description of **dumpdef** is updated to indicate that output is  
22596 written to standard error. The description of **eval** is updated to indicate that the list of excluded  
22597 C operators excludes unary ‘&’ and ‘.’. In the description of **ifdef**, the phrase “and it is not  
22598 defined to be zero” is deleted.

22599 **Issue 6**

22600 In the EXTENDED DESCRIPTION, the **eval** text is updated to include a ‘&’ character in the  
22601 excepted list.

22602 The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion  
22603 buffers.

22604 The normative text is reworded to avoid use of the term “must” for application requirements.

22605 The Open Group Base Resolution bwg2000-006 is applied.

22606 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/31 is applied, replacing the EXAMPLES  
22607 section.

22608 **NAME**

22609 mailx — process messages

22610 **SYNOPSIS**22611 **Send Mode**22612 mailx [-s *subject*] *address...*22613 **Receive Mode**

22614 mailx -e

22615 mailx [-HiNn] [-F] [-u *user*]22616 mailx -f [-HiNn] [-F] [*file*]22617 **DESCRIPTION**

22618 The *mailx* utility provides a message sending and receiving facility. It has two major modes,  
 22619 selected by the options used: Send Mode and Receive Mode.

22620 On systems that do not support the User Portability Utilities option, an application using *mailx*  
 22621 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first  
 22622 character of one or more lines is tilde ('~'), all characters in the input message shall appear in  
 22623 the delivered message, but additional characters may be inserted in the message before it is  
 22624 retrieved.

22625 On systems supporting the User Portability Utilities option, mail-receiving capabilities and other  
 22626 interactive features, Receive Mode, described below, also shall be enabled.

22627 **Send Mode**

22628 Send Mode can be used by applications or users to send messages from the text in standard  
 22629 input.

22630 **Receive Mode**

22631 Receive Mode is more oriented towards interactive users. Mail can be read and sent in this  
 22632 interactive mode.

22633 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to  
 22634 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the  
 22635 message as it is entered.

22636 Incoming mail shall be stored in one or more unspecified locations for each user, collectively  
 22637 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system  
 22638 mailbox shall be the default place to find new mail. As messages are read, they shall be marked  
 22639 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is  
 22640 called the **mbox** and is normally located in the directory referred to by the *HOME* environment  
 22641 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).  
 22642 Messages shall remain in this file until explicitly removed. When the **-f** option is used to read  
 22643 mail messages from secondary files, messages shall be retained in those files unless specifically  
 22644 removed. All three of these locations—system mailbox, **mbox**, and secondary file—are referred  
 22645 to in this section as simply “mailboxes”, unless more specific identification is required.

22646 **OPTIONS**

22647 The *mailx* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
22648 12.2, Utility Syntax Guidelines.

22649 The following options shall be supported. (Only the **–s** *subject* option shall be required on all  
22650 systems. The other options are required only on systems supporting the User Portability Utilities  
22651 option.)

22652 **–e** Test for the presence of mail in the system mailbox. The *mailx* utility shall write  
22653 nothing and exit with a successful return code if there is mail to read.

22654 **–f** Read messages from the file named by the *file* operand instead of the system  
22655 mailbox. (See also **folder**.) If no *file* operand is specified, read messages from **mbox**  
22656 instead of the system mailbox.

22657 **–F** Record the message in a file named after the first recipient. The name is the login-  
22658 name portion of the address found first on the **To:** line in the mail header.  
22659 Overrides the **record** variable, if set (see **Internal Variables in mailx** (on page  
22660 593).)

22661 **–H** Write a header summary only.

22662 **–i** Ignore interrupts. (See also **ignore**.)

22663 **–n** Do not initialize from the system default start-up file. See the EXTENDED  
22664 DESCRIPTION section.

22665 **–N** Do not write an initial header summary.

22666 **–s** *subject* Set the **Subject** header field to *subject*. All characters in the *subject* string shall  
22667 appear in the delivered message. The results are unspecified if *subject* is longer  
22668 than {LINE\_MAX} – 10 bytes or contains a <newline>.

22669 **–u** *user* Read the system mailbox of the login name *user*. This shall only be successful if  
22670 the invoking user has the appropriate privileges to read the system mailbox of that  
22671 user.

22672 **OPERANDS**

22673 The following operands shall be supported:

22674 *address* Addressee of message. When **–n** is specified and no user start-up files are accessed  
22675 (see the EXTENDED DESCRIPTION section), the user or application shall ensure  
22676 this is an address to pass to the mail delivery system. Any system or user start-up  
22677 files may enable aliases (see **alias** under **Commands in mailx** (on page 596)) that  
22678 may modify the form of *address* before it is passed to the mail delivery system.

22679 *file* A pathname of a file to be read instead of the system mailbox when **–f** is specified.  
22680 The meaning of the *file* option-argument shall be affected by the contents of the  
22681 **folder** internal variable; see **Internal Variables in mailx** (on page 593).

22682 **STDIN**

22683 When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the  
22684 message to be delivered to the specified addresses. When in Receive Mode, user commands shall  
22685 be accepted from *stdin*. If the User Portability Utilities option is not supported, standard input  
22686 lines beginning with a tilde (‘~’) character produce unspecified results.

22687 If the User Portability Utilities option is supported, then in both Send and Receive Modes,  
22688 standard input lines beginning with the escape character (usually tilde (‘~’)) shall affect  
22689 processing as described in **Command Escapes in mailx** (on page 604).

22690 **INPUT FILES**

22691 When *mailx* is used as described by this volume of IEEE Std 1003.1-2001, the *file* option-  
 22692 argument (see the **-f** option) and the **mbox** shall be text files containing mail messages,  
 22693 formatted as described in the OUTPUT FILES section. The nature of the system mailbox is  
 22694 unspecified; it need not be a file.

22695 **ENVIRONMENT VARIABLES**

22696 The following environment variables shall affect the execution of *mailx*:

22697 **DEAD** Determine the pathname of the file in which to save partial messages in case of  
 22698 interrupts or delivery errors. The default shall be **dead.letter** in the directory  
 22699 named by the *HOME* variable. The behavior of *mailx* in saving partial messages is  
 22700 unspecified if the User Portability Utilities option is not supported and *DEAD* is  
 22701 not defined with the value **/dev/null**.

22702 **EDITOR** Determine the name of a utility to invoke when the **edit** (see **Commands in mailx**  
 22703 (on page 596)) or **~e** (see **Command Escapes in mailx** (on page 604)) command is  
 22704 XSI used. The default editor is unspecified. On XSI-conformant systems it is *ed*. The  
 22705 effects of this variable are unspecified if the User Portability Utilities option is not  
 22706 supported.

22707 **HOME** Determine the pathname of the user's home directory.

22708 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 22709 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 22710 Internationalization Variables for the precedence of internationalization variables  
 22711 used to determine the values of locale categories.)

22712 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 22713 internationalization variables.

22714 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 22715 characters (for example, single-byte as opposed to multi-byte characters in  
 22716 arguments and input files) and the handling of case-insensitive address and  
 22717 header-field comparisons.

22718 **LC\_TIME** Determine the format and contents of the date and time strings written by *mailx*.

22719 **LC\_MESSAGES**

22720 Determine the locale that should be used to affect the format and contents of  
 22721 diagnostic messages written to standard error and informative messages written to  
 22722 standard output.

22723 **LISTER** Determine a string representing the command for writing the contents of the  
 22724 **folder** directory to standard output when the **folders** command is given (see  
 22725 **folders** in **Commands in mailx** (on page 596)). Any string acceptable as a  
 22726 *command\_string* operand to the *sh -c* command shall be valid. If this variable is null  
 22727 or not set, the output command shall be *ls*. The effects of this variable are  
 22728 unspecified if the User Portability Utilities option is not supported.

22729 **MAILRC** Determine the pathname of the start-up file. The default shall be **.mailrc** in the  
 22730 directory referred to by the *HOME* environment variable. The behavior of *mailx* is  
 22731 unspecified if the User Portability Utilities option is not supported and *MAILRC* is  
 22732 not defined with the value **/dev/null**.

22733 **MBOX** Determine a pathname of the file to save messages from the system mailbox that  
 22734 have been read. The **exit** command shall override this function, as shall saving the  
 22735 message explicitly in another file. The default shall be **mbox** in the directory



|       |                            |                                                                                                          |
|-------|----------------------------|----------------------------------------------------------------------------------------------------------|
| 22736 |                            | named by the <i>HOME</i> variable. The effects of this variable are unspecified if the                   |
| 22737 |                            | User Portability Utilities option is not supported.                                                      |
| 22738 | XSI                        | <b>NLSPATH</b> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .     |
| 22739 |                            |                                                                                                          |
| 22740 | <b>PAGER</b>               | Determine a string representing an output filtering or pagination command for                            |
| 22741 |                            | writing the output to the terminal. Any string acceptable as a <i>command_string</i>                     |
| 22742 |                            | operand to the <i>sh -c</i> command shall be valid. When standard output is a terminal                   |
| 22743 |                            | device, the message output shall be piped through the command if the <i>mailx</i>                        |
| 22744 |                            | internal variable <b>crt</b> is set to a value less the number of lines in the message; see              |
| 22745 |                            | <b>Internal Variables in mailx</b> (on page 593). If the <i>PAGER</i> variable is null or not set,       |
| 22746 |                            | the paginator shall be either <i>more</i> or another paginator utility documented in the                 |
| 22747 |                            | system documentation. The effects of this variable are unspecified if the User                           |
|       |                            | Portability Utilities option is not supported.                                                           |
| 22748 |                            |                                                                                                          |
| 22749 | <b>SHELL</b>               | Determine the name of a preferred command interpreter. The default shall be <i>sh</i> .                  |
| 22750 |                            | The effects of this variable are unspecified if the User Portability Utilities option is                 |
|       |                            | not supported.                                                                                           |
| 22751 |                            |                                                                                                          |
| 22752 | <b>TERM</b>                | If the internal variable <b>screen</b> is not specified, determine the name of the terminal              |
| 22753 |                            | type to indicate in an unspecified manner the number of lines in a screenful of                          |
| 22754 |                            | headers. If <i>TERM</i> is not set or is set to null, an unspecified default terminal type               |
| 22755 |                            | shall be used and the value of a screenful is unspecified. The effects of this variable                  |
|       |                            | are unspecified if the User Portability Utilities option is not supported.                               |
| 22756 |                            |                                                                                                          |
| 22757 | <b>TZ</b>                  | This variable may determine the timezone used to calculate date and time strings                         |
| 22758 |                            | written by <i>mailx</i> . If <i>TZ</i> is unset or null, an unspecified default timezone shall be        |
|       |                            | used.                                                                                                    |
| 22759 |                            |                                                                                                          |
| 22760 | <b>VISUAL</b>              | Determine a pathname of a utility to invoke when the <b>visual</b> command (see                          |
| 22761 |                            | <b>Commands in mailx</b> (on page 596)) or <b>~v</b> command-escape (see <b>Command</b>                  |
| 22762 |                            | <b>Escapes in mailx</b> (on page 604)) is used. If this variable is null or not set, the full-           |
| 22763 |                            | screen editor shall be <i>vi</i> . The effects of this variable are unspecified if the User              |
|       |                            | Portability Utilities option is not supported.                                                           |
| 22764 | <b>ASYNCHRONOUS EVENTS</b> |                                                                                                          |
| 22765 |                            | When <i>mailx</i> is in Send Mode and standard input is not a terminal, it shall take the standard       |
| 22766 |                            | action for all signals.                                                                                  |
| 22767 |                            | In Receive Mode, or in Send Mode when standard input is a terminal, if a SIGINT signal is                |
| 22768 |                            | received:                                                                                                |
| 22769 |                            | 1. If in command mode, the current command, if there is one, shall be aborted, and a                     |
| 22770 |                            | command-mode prompt shall be written.                                                                    |
| 22771 |                            | 2. If in input mode:                                                                                     |
| 22772 |                            | a. If <b>ignore</b> is set, <i>mailx</i> shall write "@\n", discard the current input line, and continue |
| 22773 |                            | processing, bypassing the message-abort mechanism described in item 2b.                                  |
| 22774 |                            | b. If the interrupt was received while sending mail, either when in Receive Mode or in                   |
| 22775 |                            | Send Mode, a message shall be written, and another subsequent interrupt, with no                         |
| 22776 |                            | other intervening characters typed, shall be required to abort the mail message. If in                   |
| 22777 |                            | Receive Mode and another interrupt is received, a command-mode prompt shall be                           |
| 22778 |                            | written. If in Send Mode and another interrupt is received, <i>mailx</i> shall terminate with            |
| 22779 |                            | a non-zero status.                                                                                       |
| 22780 |                            | In both cases listed in item b, if the message is not empty:                                             |

22781                   i. If **save** is enabled and the file named by *DEAD* can be created, the message  
 22782 shall be written to the file named by *DEAD*. If the file exists, the message shall  
 22783 be written to replace the contents of the file.

22784                   ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the  
 22785 message shall not be saved.

22786           The *mailx* utility shall take the standard action for all other signals.

## 22787 **STDOUT**

22788           In command and input modes, all output, including prompts and messages, shall be written to  
 22789 standard output.

## 22790 **STDERR**

22791           The standard error shall be used only for diagnostic messages.

## 22792 **OUTPUT FILES**

22793           Various *mailx* commands and command escapes can create or add to files, including the **mbox**,  
 22794 the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of  
 22795 IEEE Std 1003.1-2001, these files shall be text files, formatted as follows:

22796           line beginning with **From**<space>  
 22797           [one or more *header-lines*; see **Commands in mailx** (on page 596)]  
 22798           empty line  
 22799           [zero or more *body lines*  
 22800           empty line]  
 22801           [line beginning with **From**<space>...]

22802           where each message begins with the **From** <space> line shown, preceded by the beginning of  
 22803 the file or an empty line. (The **From** <space> line is considered to be part of the message header,  
 22804 but not one of the header-lines referred to in **Commands in mailx** (on page 596); thus, it shall not  
 22805 be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the  
 22806 **From** <space> line and any additional header lines are unspecified, except that none shall be  
 22807 empty. The format of a message body line is also unspecified, except that no line following an  
 22808 empty line shall start with **From** <space>; *mailx* shall modify any such user-entered message  
 22809 body lines (following an empty line and beginning with **From** <space>) by adding one or more  
 22810 characters to precede the 'F'; it may add these characters to **From** <space> lines that are not  
 22811 preceded by an empty line.

22812           When a message from the system mailbox or entered by the user is not a text file, it is  
 22813 implementation-defined how such a message is stored in files written by *mailx*.

## 22814 **EXTENDED DESCRIPTION**

22815           The entire EXTENDED DESCRIPTION section shall apply only to implementations supporting  
 22816 the User Portability Utilities option.

22817           The *mailx* utility cannot guarantee support for all character encodings in all circumstances. For  
 22818 example, inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data  
 22819 need not be portable to non-internationalized systems, and so on. Under these circumstances, it  
 22820 is recommended that only characters defined in the ISO/IEC 646:1991 standard International  
 22821 Reference Version (equivalent to ASCII) 7-bit range of characters be used.

22822           When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of  
 22823 header-summary lines (if **-N** was not specified and there are messages, see below), followed by  
 22824 a prompt indicating that *mailx* can accept regular commands (see **Commands in mailx** (on page  
 22825 596)); this is termed *command mode*. The page of header-summary lines shall contain the first  
 22826 new message if there are new messages, or the first unread message if there are unread  
 22827 messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and

standard input is a terminal, if no subject is specified on the command line and the **asksub** variable is set, a prompt for the subject shall be written. At this point, *mailx* shall be in input mode. This input mode shall also be entered when using one of the Receive Mode synopsis forms and a reply or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or **mail** commands and standard input is a terminal. When the message is typed and the end of the message is encountered, the message shall be passed to the mail delivery software. Commands can be entered by beginning a line with the escape character (by default, tilde ('~')) followed by a single command letter and optional arguments. See **Commands in mailx** (on page 596) for a summary of these commands. It is unspecified what effect these commands will have if standard input is not a terminal when a message is entered using either the Send Mode synopsis, or the Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

**Note:** For notational convenience, this section uses the default escape character, tilde, in all references and examples.

At any time, the behavior of *mailx* shall be governed by a set of environmental and internal variables. These are flags and valued parameters that can be set and cleared via the *mailx* **set** and **unset** commands.

Regular commands are of the form:

[*command*] [*msglist*] [*argument ...*]

If no *command* is specified in command mode, **next** shall be assumed. In input mode, commands shall be recognized by the escape character, and lines not treated as commands shall be taken as input for the message.

In command mode, each message shall be assigned a sequential number, starting with 1.

All messages have a state that shall affect how they are displayed in the header summary and how they are retained or deleted upon termination of *mailx*. There is at any time the notion of a *current* message, which shall be marked by a '>' at the beginning of a line in the header summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current message shall be the first new message, if there is a new message, or the first unread message if there is an unread message, or the first message if there are any messages, or unspecified if there are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*) or an optional single message (*message*) on which to operate shall leave the current message set to the highest-numbered message of the messages specified, unless the command deletes messages, in which case the current message shall be set to the first undeleted message (that is, a message not in the deleted state) after the highest-numbered message deleted by the command, if one exists, or the first undeleted message before the highest-numbered message deleted by the command, if one exists, or to an unspecified value if there are no remaining undeleted messages. All messages shall be in one of the following states:

*new* The message is present in the system mailbox and has not been viewed by the user or moved to any other state. Messages in state *new* when *mailx* quits shall be retained in the system mailbox.

*unread* The message has been present in the system mailbox for more than one invocation of *mailx* and has not been viewed by the user or moved to any other state. Messages in state *unread* when *mailx* quits shall be retained in the system mailbox.

*read* The message has been processed by one of the following commands: **~f**, **~m**, **~F**, **~M**, **copy**, **mbox**, **next**, **pipe**, **print**, **Print**, **top**, **type**, **Type**, **undelete**. The **delete**, **dp**, and **dt** commands may also cause the next message to be marked as *read*, depending on the value of the **autoprint** variable. Messages that are in the system mailbox and in state *read* when *mailx* quits shall be saved in the **mbox**, unless the internal variable **hold** was set. Messages that are in the **mbox** or in a secondary mailbox and in state

|       |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 22876 |                  | <i>read</i> when <i>mailx</i> quits shall be retained in their current location.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 22877 | <i>deleted</i>   | The message has been processed by one of the following commands: <b>delete</b> , <b>dp</b> , <b>dt</b> . Messages in state <i>deleted</i> when <i>mailx</i> quits shall be deleted. Deleted messages shall be ignored until <i>mailx</i> quits or changes mailboxes or they are specified to the undelete command; for example, the message specification <i>/string</i> shall only search the subject lines of messages that have not yet been deleted, unless the command operating on the list of messages is <b>undelete</b> . No deleted message or deleted message header shall be displayed by any <i>mailx</i> command other than <b>undelete</b> . |
| 22878 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22879 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22880 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22881 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22882 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22883 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22884 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22885 | <i>preserved</i> | The message has been processed by a <b>preserve</b> command. When <i>mailx</i> quits, the message shall be retained in its current location.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 22886 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22887 | <i>saved</i>     | The message has been processed by one of the following commands: <b>save</b> or <b>write</b> . If the current mailbox is the system mailbox, and the internal variable <b>keepsave</b> is set, messages in the state <i>saved</i> shall be saved to the file designated by the <i>MBOX</i> variable (see the ENVIRONMENT VARIABLES section). If the current mailbox is the system mailbox, messages in the state <i>saved</i> shall be deleted from the current mailbox, when the <b>quit</b> or <b>file</b> command is used to exit the current mailbox.                                                                                                   |
| 22888 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22889 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22890 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22891 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22892 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22893 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22894 |                  | The header-summary line for each message shall indicate the state of the message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 22895 |                  | Many commands take an optional list of messages ( <i>msglist</i> ) on which to operate, which defaults to the current message. A <i>msglist</i> is a list of message specifications separated by <blank>s, which can include:                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 22896 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22897 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22898 | <i>n</i>         | Message number <i>n</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 22899 | <i>+</i>         | The next undeleted message, or the next deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 22900 | <i>–</i>         | The next previous undeleted message, or the next previous deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 22901 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22902 | <i>.</i>         | The current message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22903 | <i>^</i>         | The first undeleted message, or the first deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 22904 | <i>\$</i>        | The last message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 22905 | <i>*</i>         | All messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 22906 | <i>n–m</i>       | An inclusive range of message numbers.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 22907 | <i>address</i>   | All messages from <i>address</i> ; any address as shown in a header summary shall be matchable in this form.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 22908 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22909 | <i>/string</i>   | All messages with <i>string</i> in the subject line (case ignored).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22910 | <i>:c</i>        | All messages of type <i>c</i> , where <i>c</i> shall be one of:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 22911 | <i>d</i>         | Deleted messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 22912 | <i>n</i>         | New messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 22913 | <i>o</i>         | Old messages (any not in state <i>read</i> or <i>new</i> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 22914 | <i>r</i>         | Read messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 22915 | <i>u</i>         | Unread messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

Other commands take an optional message (*message*) on which to operate, which defaults to the current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more than one message is specified, only the first shall be operated on.

Other arguments are usually arbitrary strings whose usage depends on the command involved.

## Start-Up in mailx

At start-up time, *mailx* shall take the following steps in sequence:

1. Establish all variables at their stated default values.
2. Process command line options, overriding corresponding default values.
3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables that are present in the environment, overriding the corresponding default values.
4. Read *mailx* commands from an unspecified system start-up file, unless the **-n** option is given, to initialize any internal *mailx* variables and aliases.
5. Process the start-up file of *mailx* commands named in the user *MAILRC* variable.

Most regular *mailx* commands are valid inside start-up files, the most common use being to set up initial display options and alias lists. The following commands shall be invalid in the start-up file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, **visual**, **Copy**, **followup**, and **Followup**. Any errors in the start-up file shall either cause *mailx* to terminate with a diagnostic message and a non-zero status or to continue after writing a diagnostic message, ignoring the remainder of the lines in the start-up file.

A blank line in a start-up file shall be ignored.

## Internal Variables in mailx

The following variables are internal *mailx* variables. Each internal variable can be set via the *mailx* **set** command at any time. The **unset** and **set no name** commands can be used to erase variables.

In the following list, variables shown as:

variable

represent Boolean values. Variables shown as:

variable=value

shall be assigned string or numeric values. For string values, the rules in **Commands in mailx** (on page 596) concerning filenames and quoting shall also apply.

The defaults specified here may be changed by the implementation-defined system start-up file unless the user specifies the **-n** option.

**allnet** All network names whose login name components match shall be treated as identical. This shall cause the *msglist* message specifications to behave similarly. The default shall be **noallnet**. See also the **alternates** command and the **metoo** variable.

**append** Append messages to the end of the **mbox** file upon termination instead of placing them at the beginning. The default shall be **noappend**. This variable shall not affect the **save** command when saving to **mbox**.

**ask, asksub** Prompt for a subject line on outgoing mail if one is not specified on the command line with the **-s** option. The **ask** and **asksub** forms are synonyms; the system shall

|           |                         |                                                                                                             |
|-----------|-------------------------|-------------------------------------------------------------------------------------------------------------|
| 22957     |                         | refer to <b>asksub</b> and <b>noasksub</b> in its messages, but shall accept <b>ask</b> and <b>noask</b> as |
| 22958     |                         | user input to mean <b>asksub</b> and <b>noasksub</b> . It shall not be possible to set both <b>ask</b>      |
| 22959     |                         | and <b>noasksub</b> , or <b>noask</b> and <b>asksub</b> . The default shall be <b>asksub</b> , but no       |
| 22960     |                         | prompting shall be done if standard input is not a terminal.                                                |
| 22961     | <b>askbcc</b>           | Prompt for the blind copy list. The default shall be <b>noaskbcc</b> .                                      |
| 22962     | <b>askcc</b>            | Prompt for the copy list. The default shall be <b>noaskcc</b> .                                             |
| 22963     | <b>autoprint</b>        | Enable automatic writing of messages after <b>delete</b> and <b>undelete</b> commands. The                  |
| 22964     |                         | default shall be <b>noautoprint</b> .                                                                       |
| 22965     | <b>bang</b>             | Enable the special-case treatment of exclamation marks ('!') in escape command                              |
| 22966     |                         | lines; see the <b>escape</b> command and <b>Command Escapes in mailx</b> (on page 604).                     |
| 22967     |                         | The default shall be <b>nobang</b> , disabling the expansion of '!' in the <i>command</i>                   |
| 22968     |                         | argument to the ~! command and the ~<!command escape.                                                       |
| 22969     | <b>cmd=command</b>      |                                                                                                             |
| 22970     |                         | Set the default command to be invoked by the <b>pipe</b> command. The default shall be                      |
| 22971     |                         | <b>nocmd</b> .                                                                                              |
| 22972     | <b>crt=number</b>       | Pipe messages having more than <i>number</i> lines through the command specified by                         |
| 22973     |                         | the value of the <i>PAGER</i> variable. The default shall be <b>nocrt</b> . If it is set to null, the       |
| 22974     |                         | value used is implementation-defined.                                                                       |
| 22975 XSI | <b>debug</b>            | Enable verbose diagnostics for debugging. Messages are not delivered. The                                   |
| 22976     |                         | default shall be <b>nodebug</b> .                                                                           |
| 22977     | <b>dot</b>              | When <b>dot</b> is set, a period on a line by itself during message input from a terminal                   |
| 22978     |                         | shall also signify end-of-file (in addition to normal end-of-file). The default shall be                    |
| 22979     |                         | <b>nodot</b> . If <b>ignoreeof</b> is set (see below), a setting of <b>nodot</b> shall be ignored and the   |
| 22980     |                         | period is the only method to terminate input mode.                                                          |
| 22981     | <b>escape=c</b>         | Set the command escape character to be the character 'c'. By default, the                                   |
| 22982     |                         | command escape character shall be tilde. If <b>escape</b> is unset, tilde shall be used; if it              |
| 22983     |                         | is set to null, command escaping shall be disabled.                                                         |
| 22984     | <b>flipr</b>            | Reverse the meanings of the <b>R</b> and <b>r</b> commands. The default shall be <b>noflipr</b> .           |
| 22985     | <b>folder=directory</b> |                                                                                                             |
| 22986     |                         | The default directory for saving mail files. User-specified filenames beginning with                        |
| 22987     |                         | a plus sign ('+') shall be expanded by preceding the filename with this directory                           |
| 22988     |                         | name to obtain the real pathname. If <i>directory</i> does not start with a slash ('/'), the                |
| 22989     |                         | contents of <i>HOME</i> shall be prefixed to it. The default shall be <b>nofolder</b> . If <b>folder</b> is |
| 22990     |                         | unset or set to null, user-specified filenames beginning with '+' shall refer to files                      |
| 22991     |                         | in the current directory that begin with the literal '+' character. See also <b>outfolder</b>               |
| 22992     |                         | below. The <b>folder</b> value need not affect the processing of the files named in <i>MBOX</i>             |
| 22993     |                         | and <i>DEAD</i> .                                                                                           |
| 22994     | <b>header</b>           | Enable writing of the header summary when entering <i>mailx</i> in Receive Mode. The                        |
| 22995     |                         | default shall be <b>header</b> .                                                                            |
| 22996     | <b>hold</b>             | Preserve all messages that are read in the system mailbox instead of putting them                           |
| 22997     |                         | in the <b>mbox</b> save file. The default shall be <b>nohold</b> .                                          |
| 22998     | <b>ignore</b>           | Ignore interrupts while entering messages. The default shall be <b>noignore</b> .                           |
| 22999     | <b>ignoreeof</b>        | Ignore normal end-of-file during message input. Input can be terminated only by                             |
| 23000     |                         | entering a period ('.') on a line by itself or by the ~. command escape. The default                        |
| 23001     |                         | shall be <b>noignoreeof</b> . See also <b>dot</b> above.                                                    |

|           |                            |                                                                                                            |
|-----------|----------------------------|------------------------------------------------------------------------------------------------------------|
| 23002     | <b>indentprefix=string</b> |                                                                                                            |
| 23003     |                            | A string that shall be added as a prefix to each line that is inserted into the message                    |
| 23004     |                            | by the <b>~m</b> command escape. This variable shall default to one <tab>.                                 |
| 23005     | <b>keep</b>                | When a system mailbox, secondary mailbox, or <b>mbox</b> is empty, truncate it to zero                     |
| 23006     |                            | length instead of removing it. The default shall be <b>nokeep</b> .                                        |
| 23007     | <b>keepsave</b>            | Keep the messages that have been saved from the system mailbox into other files                            |
| 23008     |                            | in the file designated by the variable <i>MBOX</i> , instead of deleting them. The default                 |
| 23009     |                            | shall be <b>nokeepsave</b> .                                                                               |
| 23010     | <b>metoo</b>               | Suppress the deletion of the login name of the user from the recipient list when                           |
| 23011     |                            | replying to a message or sending to a group. The default shall be <b>nometoo</b> .                         |
| 23012 XSI | <b>onehop</b>              | When responding to a message that was originally sent to several recipients, the                           |
| 23013     |                            | other recipient addresses are normally forced to be relative to the originating                            |
| 23014     |                            | author's machine for the response. This flag disables alteration of the recipients'                        |
| 23015     |                            | addresses, improving efficiency in a network where all machines can send directly                          |
| 23016     |                            | to all other machines (that is, one hop away). The default shall be <b>noonehop</b> .                      |
| 23017     | <b>outfolder</b>           | Cause the files used to record outgoing messages to be located in the directory                            |
| 23018     |                            | specified by the <b>folder</b> variable unless the pathname is absolute. The default shall                 |
| 23019     |                            | be <b>nooutfolder</b> . See the <b>record</b> variable.                                                    |
| 23020     | <b>page</b>                | Insert a <form-feed> after each message sent through the pipe created by the <b>pipe</b>                   |
| 23021     |                            | command. The default shall be <b>nopage</b> .                                                              |
| 23022     | <b>prompt=string</b>       |                                                                                                            |
| 23023     |                            | Set the command-mode prompt to <i>string</i> . If <i>string</i> is null or if <b>noprompt</b> is set, no   |
| 23024     |                            | prompting shall occur. The default shall be to prompt with the string " ? ".                               |
| 23025     | <b>quiet</b>               | Refrain from writing the opening message and version when entering <i>mailx</i> . The                      |
| 23026     |                            | default shall be <b>noquiet</b> .                                                                          |
| 23027     | <b>record=file</b>         | Record all outgoing mail in the file with the pathname <i>file</i> . The default shall be                  |
| 23028     |                            | <b>no record</b> . See also <b>outfolder</b> above.                                                        |
| 23029     | <b>save</b>                | Enable saving of messages in the dead-letter file on interrupt or delivery error. See                      |
| 23030     |                            | the variable <i>DEAD</i> for the location of the dead-letter file. The default shall be <b>save</b> .      |
| 23031     | <b>screen=number</b>       |                                                                                                            |
| 23032     |                            | Set the number of lines in a screenful of headers for the <b>headers</b> and <b>z</b> commands.            |
| 23033     |                            | If <b>screen</b> is not specified, a value based on the terminal type identified by the                    |
| 23034     |                            | <i>TERM</i> environment variable, the window size, the baud rate, or some combination                      |
| 23035     |                            | of these shall be used.                                                                                    |
| 23036     | <b>sendwait</b>            | Wait for the background mailer to finish before returning. The default shall be                            |
| 23037     |                            | <b>nosendwait</b> .                                                                                        |
| 23038     | <b>showto</b>              | When the sender of the message was the user who is invoking <i>mailx</i> , write the                       |
| 23039     |                            | information from the <b>To:</b> line instead of the <b>From:</b> line in the header summary.               |
| 23040     |                            | The default shall be <b>noshowto</b> .                                                                     |
| 23041     | <b>sign=string</b>         | Set the variable inserted into the text of a message when the <b>~a</b> command escape is                  |
| 23042     |                            | given. The default shall be <b>no sign</b> . The character sequences ' <b>\t</b> ' and ' <b>\n</b> ' shall |
| 23043     |                            | be recognized in the variable as <tab>s and <newline>s, respectively. (See also <b>~i</b> in               |
| 23044     |                            | <b>Command Escapes in mailx</b> (on page 604).)                                                            |
| 23045     | <b>Sign=string</b>         | Set the variable inserted into the text of a message when the <b>~A</b> command escape is                  |
| 23046     |                            | given. The default shall be <b>no Sign</b> . The character sequences ' <b>\t</b> ' and ' <b>\n</b> ' shall |

23047 be recognized in the variable as <tab>s and <newline>s, respectively.

23048 **toplines**=*number*

23049 Set the number of lines of the message to write with the **top** command. The default  
23050 shall be 5.

## 23051 **Commands in mailx**

23052 The following *mailx* commands shall be provided. In the following list, header refers to lines  
23053 from the message header, as shown in the OUTPUT FILES section. Header-line refers to lines  
23054 within the header that begin with one or more non-white-space characters, immediately  
23055 followed by a colon and white space and continuing until the next line beginning with a non-  
23056 white-space character or an empty line. Header-field refers to the portion of a header line prior  
23057 to the first colon in that line.

23058 For each of the commands listed below, the command can be entered as the abbreviation (those  
23059 characters in the Synopsis command word preceding the ' [ ' ), the full command (all characters  
23060 shown for the command word, omitting the ' [ ' and ' ] ' ), or any truncation of the full  
23061 command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the  
23062 Synopsis) can be entered as **ex**, **exi**, or **exit**.

23063 The arguments to commands can be quoted, using the following methods:

- 23064 • An argument can be enclosed between paired double-quotes ( " ") or single-quotes ( ' '); any  
23065 white space, shell word expansion, or backslash characters within the quotes shall be treated  
23066 literally as part of the argument. A double-quote shall be treated literally within single-  
23067 quotes and *vice versa*. These special properties of the quote marks shall occur only when they  
23068 are paired at the beginning and end of the argument.
- 23069 • A backslash outside of the enclosing quotes shall be discarded and the following character  
23070 treated literally as part of the argument.
- 23071 • An unquoted backslash at the end of a command line shall be discarded and the next line  
23072 shall continue the command.

23073 Filenames, where expected, shall be subjected to the following transformations, in sequence:

- 23074 • If the filename begins with an unquoted plus sign, and the **folder** variable is defined (see the  
23075 **folder** variable), the plus sign shall be replaced by the value of the **folder** variable followed  
23076 by a slash. If the **folder** variable is unset or is set to null, the filename shall be unchanged.
- 23077 • Shell word expansions shall be applied to the filename (see Section 2.6 (on page 36)). If more  
23078 than a single pathname results from this expansion and the command is expecting one file,  
23079 the effects are unspecified.

## 23080 **Declare Aliases**

23081 *Synopsis:*     a[lias] [alias [address...]]

23082               g[roup] [alias [address...]]

23083 Add the given addresses to the alias specified by *alias*. The names shall be substituted when  
23084 *alias* is used as a recipient address specified by the user in an outgoing message (that is, other  
23085 recipients addressed indirectly through the **reply** command shall not be substituted in this  
23086 manner). Mail address alias substitution shall apply only when the alias string is used as a full  
23087 address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution. If  
23088 no arguments are given, write a listing of the current aliases to standard output. If only an *alias*  
23089 argument is given, write a listing of the specified alias to standard output. These listings need  
23090 not reflect the same order of addresses that were entered.



23091 **Declare Alternatives**23092 *Synopsis:*     alt[ernates] *name...*

23093 (See also the **metoo** command.) Declare a list of alternative names for the user's login. When  
 23094 responding to a message, these names shall be removed from the list of recipients for the  
 23095 response. The comparison of names shall be in a case-insensitive manner. With no arguments,  
 23096 **alternates** shall write the current list of alternative names.

23097 **Change Current Directory**

23098 *Synopsis:*     cd [*directory*]  
 23099                 ch[dir] [*directory*]

23100 Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

23101 **Copy Messages**

23102 *Synopsis:*     c[opy] [*file*]  
 23103                 c[opy] [*msglist*] *file*  
 23104                 C[opy] [*msglist*]

23105 Copy messages to the file named by the pathname *file* without marking the messages as saved.  
 23106 Otherwise, it shall be equivalent to the **save** command.

23107 In the capitalized form, save the specified messages in a file whose name is derived from the  
 23108 author of the message to be saved, without marking the messages as saved. Otherwise, it shall  
 23109 be equivalent to the **Save** command.

23110 **Delete Messages**23111 *Synopsis:*     d[ele]te [*msglist*]

23112 Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see  
 23113 the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there  
 23114 are messages remaining after the **delete** command, the current message shall be written as  
 23115 described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be  
 23116 written.

23117 **Discard Header Fields**

23118 *Synopsis:*     di[scard] [*header-field...*]  
 23119                 ig[nore] [*header-field...*]

23120 Suppress the specified header fields when writing messages. Specified *header-fields* shall be  
 23121 added to the list of suppressed header fields. Examples of header fields to ignore are **status** and  
 23122 **cc**. The fields shall be included when the message is saved. The **Print** and **Type** commands shall  
 23123 override this command. The comparison of header fields shall be in a case-insensitive manner. If  
 23124 no arguments are specified, write a list of the currently suppressed header fields to standard  
 23125 output; the listing need not reflect the same order of header fields that were entered.

23126 If both **retain** and **discard** commands are given, **discard** commands shall be ignored.

## 23127 Delete Messages and Display

23128 Synopsis: dp [msglist]  
 23129 dt [msglist]

23130 Delete the specified messages as described for the **delete** command, except that the **autoprint**  
 23131 variable shall have no effect, and the current message shall be written only if it was set to a  
 23132 message after the last message deleted by the command. Otherwise, an informational message  
 23133 to the effect that there are no further messages in the mailbox shall be written, followed by the  
 23134 *mailx* prompt.

## 23135 Echo a String

23136 Synopsis: ec[ho] string ...

23137 Echo the given strings, equivalent to the shell *echo* utility.

## 23138 Edit Messages

23139 Synopsis: e[dit] [msglist]

23140 Edit the given messages. The messages shall be placed in a temporary file and the utility named  
 23141 by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is  
 23142 unspecified.

23143 The **edit** command does not modify the contents of those messages in the mailbox.

## 23144 Exit

23145 Synopsis: ex[it]  
 23146 x[it]

23147 Exit from *mailx* without changing the mailbox. No messages shall be saved in the **mbox** (see also  
 23148 **quit**).

## 23149 Change Folder

23150 Synopsis: fi[le] [file]  
 23151 fold[er] [file]

23152 Quit (see the **quit** command) from the current file of messages and read in the file named by the  
 23153 pathname *file*. If no argument is given, the name and status of the current mailbox shall be  
 23154 written.

23155 Several unquoted special characters shall be recognized when used as *file* names, with the  
 23156 following substitutions:

23157 % The system mailbox for the invoking user.

23158 %*user* The system mailbox for *user*.

23159 # The previous file.

23160 & The current **mbox**.

23161 +*file* The named file in the **folder** directory. (See the **folder** variable.)

23162 The default file shall be the current mailbox.

## 23163 **Display List of Folders**

23164 *Synopsis:*     *folders*

23165 Write the names of the files in the directory set by the **folder** variable. The command specified by  
23166 the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES section).

## 23167 **Follow Up Specified Messages**

23168 *Synopsis:*     *fo[llowup] [message]*

23169               *F[ollowup] [msglist]*

23170 In the lowercase form, respond to a message, recording the response in a file whose name is  
23171 derived from the author of the message. See also the **save** and **copy** commands and **outfolder**.

23172 In the capitalized form, respond to the first message in the *msglist*, sending the message to the  
23173 author of each message in the *msglist*. The subject line shall be taken from the first message and  
23174 the response shall be recorded in a file whose name is derived from the author of the first  
23175 message. See also the **Save** and **Copy** commands and **outfolder**.

23176 Both forms shall override the **record** variable, if set.

## 23177 **Display Header Summary for Specified Messages**

23178 *Synopsis:*     *f[rom] [msglist]*

23179 Write the header summary for the specified messages.

## 23180 **Display Header Summary**

23181 *Synopsis:*     *h[eaders] [message]*

23182 Write the page of headers that includes the message specified. If the *message* argument is not  
23183 specified, the current message shall not change. However, if the *message* argument is specified,  
23184 the current message shall become the message that appears at the top of the page of headers that  
23185 includes the message specified. The **screen** variable sets the number of headers per page. See  
23186 also the **z** command.

## 23187 **Help**

23188 *Synopsis:*     *hel[p]*

23189               ?

23190 Write a summary of commands.

## 23191 **Hold Messages**

23192 *Synopsis:*     *ho[ld] [msglist]*

23193               *pre[serve] [msglist]*

23194 Mark the messages in *msglist* to be retained in the mailbox when *mailx* terminates. This shall  
23195 override any commands that might previously have marked the messages to be deleted. During  
23196 the current invocation of *mailx*, only the **delete**, **dp**, or **dt** commands shall remove the *preserve*  
23197 marking of a message.

**Execute Commands Conditionally**

Synopsis:    i[f] s|r  
               mail-commands  
               el[se]  
               mail-commands  
               en[dif]

Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be executed only in Receive Mode.

**List Available Commands**

Synopsis:    l[ist]

Write a list of all commands available. No explanation shall be given.

**Mail a Message**

Synopsis:    m[ail] address...

Mail a message to the specified addresses or aliases.

**Direct Messages to mbox**

Synopsis:    mb[ox] [msglist]

Arrange for the given messages to end up in the **mbox** save file when *mailx* terminates normally. See *MBOX*. See also the **exit** and **quit** commands.

**Process Next Specified Message**

Synopsis:    n[ext] [message]

If the current message has not been written (for example, by the **print** command) since *mailx* started or since any other message was the current message, behave as if the **print** command was entered. Otherwise, if there is an undeleted message after the current message, make it the current message and behave as if the **print** command was entered. Otherwise, an informational message to the effect that there are no further messages in the mailbox shall be written, followed by the *mailx* prompt.

**Pipe Message**

Synopsis:    pi[pe] [[msglist] command]  
               | [[msglist] command]

Pipe the messages through the given *command* by invoking the command interpreter specified by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) The application shall ensure that the command is given as a single argument. Quoting, described previously, can be used to accomplish this. If no arguments are given, the current message shall be piped through the command specified by the value of the **cmd** variable. If the **page** variable is set, a <form-feed> shall be inserted after each message.

### 23234 **Display Message with Headers**

23235 *Synopsis:*   P[rint] [msglist]  
 23236           T[ype] [msglist]

23237 Write the specified messages, including all header lines, to standard output. Override  
 23238 suppression of lines by the **discard**, **ignore**, and **retain** commands. If **crt** is set, the messages  
 23239 longer than the number of lines specified by the **crt** variable shall be paged through the  
 23240 command specified by the *PAGER* environment variable.

### 23241 **Display Message**

23242 *Synopsis:*   p[rint] [msglist]  
 23243           t[ype] [msglist]

23244 Write the specified messages to standard output. If **crt** is set, the messages longer than the  
 23245 number of lines specified by the **crt** variable shall be paged through the command specified by  
 23246 the *PAGER* environment variable.

### 23247 **Quit**

23248 *Synopsis:*   q[uit]  
 23249           end-of-file

23250 Terminate *mailx*, storing messages that were read in **mbox** (if the current mailbox is the system  
 23251 mailbox and unless **hold** is set), deleting messages that have been explicitly saved (unless  
 23252 **keepsave** is set), discarding messages that have been deleted, and saving all remaining messages  
 23253 in the mailbox.

### 23254 **Reply to a Message List**

23255 *Synopsis:*   R[eply] [msglist]  
 23256           R[espond] [msglist]

23257 Mail a reply message to the sender of each message in the *msglist*. The subject line shall be  
 23258 formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject  
 23259 from the first message. If **record** is set to a filename, the response shall be saved at the end of that  
 23260 file.

23261 See also the **flipr** variable.

### 23262 **Reply to a Message**

23263 *Synopsis:*   r[eply] [message]  
 23264           r[espond] [message]

23265 Mail a reply message to all recipients included in the header of the message. The subject line  
 23266 shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the  
 23267 subject from the message. If **record** is set to a filename, the response shall be saved at the end of  
 23268 that file.

23269 See also the **flipr** variable.

**Retain Header Fields**

Synopsis: `ret[ain] [header-field...]`

Retain the specified header fields when writing messages. This command shall override all **discard** and **ignore** commands. The comparison of header fields shall be in a case-insensitive manner. If no arguments are specified, write a list of the currently retained header fields to standard output; the listing need not reflect the same order of header fields that were entered.

**Save Messages**

Synopsis: `s[ave] [file]`  
`s[ave] [msglist] file`  
`S[ave] [msglist]`

Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file* argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be appended to the file. The message shall be put in the state *saved*, and shall behave as specified in the description of the *saved* state when the current mailbox is exited by the **quit** or **file** command.

In the capitalized form, save the specified messages in a file whose name is derived from the author of the first message. The name of the file shall be taken to be the author's name with all network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and **outfolder** variable.

**Set Variables**

Synopsis: `se[t] [name=[string]] ... [name=number ...] [noname ...]`

Define one or more variables called *name*. The variable can be given a null, string, or numeric value. Quoting and backslash escapes can occur anywhere in *string*, as described previously, as if the *string* portion of the argument were the entire argument. The forms *name* and *name=* shall be equivalent to *name=""* for variables that take string values. The **set** command without arguments shall write a list of all defined variables and their values. The **no name** form shall be equivalent to **unset name**.

**Invoke a Shell**

Synopsis: `sh[ell]`

Invoke an interactive command interpreter (see also *SHELL*).

**Display Message Size**

Synopsis: `si[ze] [msglist]`

Write the size in bytes of each of the specified messages.

**Read mailx Commands From a File**

Synopsis: `so[urce] file`

Read and execute commands from the file named by the pathname *file* and return to command mode.

### 23307 **Display Beginning of Messages**

23308 *Synopsis:*    to[p] [msglist]

23309 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken  
23310 as the number of lines to write. The default shall be 5.

### 23311 **Touch Messages**

23312 *Synopsis:*    tou[ch] [msglist]

23313 Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a  
23314 file, it shall be placed in the **mbx** upon normal termination. See **exit** and **quit**.

### 23315 **Delete Aliases**

23316 *Synopsis:*    una[lias] [alias]...

23317 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

### 23318 **Undelete Messages**

23319 *Synopsis:*    u[ndelete] [msglist]

23320 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last  
23321 message of those restored shall be written. If *msglist* is not specified, the message shall be  
23322 selected as follows:

- 23323       • If there are any deleted messages that follow the current message, the first of these shall be  
23324       chosen.
- 23325       • Otherwise, the last deleted message that also precedes the current message shall be chosen.

### 23326 **Unset Variables**

23327 *Synopsis:*    uns[et] name...

23328 Cause the specified variables to be erased.

### 23329 **Edit Message with Full-Screen Editor**

23330 *Synopsis:*    v[isual] [msglist]

23331 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,  
23332 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The  
23333 default editor shall be *vi*.

23334 The **visual** command does not modify the contents of those messages in the mailbox.

### 23335 **Write Messages to a File**

23336 *Synopsis:*    w[rite] [msglist] file

23337 Write the given messages to the file specified by the pathname *file*, minus the message header.  
23338 Otherwise, it shall be equivalent to the **save** command.

**Scroll Header Display**

Synopsis:     z [+|-]

Scroll the header display forward (if '+' is specified or if no option is specified) or backward (if '-' is specified) one screenful. The number of headers written shall be set by the **screen** variable.

**Invoke Shell Command**

Synopsis:     ! *command*

Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall be replaced with the command executed by the previous ! command or ~! command escape.

**Null Command**

Synopsis:     # *comment*

This null command (comment) shall be ignored by *mailx*.

**Display Current Message Number**

Synopsis:     =

Write the current message number.

**Command Escapes in mailx**

The following commands can be entered only from input mode, by beginning a line with the escape character (by default, tilde ('~')). See the **escape** variable description for changing this special character. The format for the commands shall be:

<escape-character><command-char><separator> [<arguments>]

where the <separator> can be zero or more <blank>s.

In the following descriptions, the application shall ensure that the argument *command* (but not *mailx-command*) is a shell command string. Any string acceptable to the command interpreter specified by the *SHELL* variable when it is invoked as *SHELL -c command\_string* shall be valid. The command can be presented as multiple arguments (that is, quoting is not required).

Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send Mode and produce unspecified results.

~! *command*   Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*; and then return to input mode. If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall be replaced with the command executed by the previous ! command or ~! command escape.

~.            Simulate end-of-file (terminate message input).

~: *mailx-command*, ~\_ *mailx-command*  
Perform the command-level request.

~?           Write a summary of command escapes.

~A           This shall be equivalent to ~i **Sign**.

~a           This shall be equivalent to ~i **sign**.



|       |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
|-------|-------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 23377 | <b>~b</b> <i>name...</i>                                                                                    | Add the <i>names</i> to the blind carbon copy ( <b>Bcc</b> ) list.                                                                                                                                                                                                                                                                                                                    |
| 23378 | <b>~c</b> <i>name...</i>                                                                                    | Add the <i>names</i> to the carbon copy ( <b>Cc</b> ) list.                                                                                                                                                                                                                                                                                                                           |
| 23379 | <b>~d</b>                                                                                                   | Read in the dead-letter file. See <i>DEAD</i> for a description of this file.                                                                                                                                                                                                                                                                                                         |
| 23380 | <b>~e</b>                                                                                                   | Invoke the editor, as specified by the <i>EDITOR</i> environment variable, on the partial message.                                                                                                                                                                                                                                                                                    |
| 23381 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23382 | <b>~f</b> [ <i>msglist</i> ]                                                                                | Forward the specified messages. The specified messages shall be inserted into the current message without alteration. This command escape also shall insert message headers into the message with field selection affected by the <b>discard</b> , <b>ignore</b> , and <b>retain</b> commands.                                                                                        |
| 23383 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23384 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23385 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23386 | <b>~F</b> [ <i>msglist</i> ]                                                                                | This shall be the equivalent of the <b>~f</b> command escape, except that all headers shall be included in the message, regardless of previous <b>discard</b> , <b>ignore</b> , and <b>retain</b> commands.                                                                                                                                                                           |
| 23387 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23388 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23389 | <b>~h</b>                                                                                                   | If standard input is a terminal, prompt for a <b>Subject</b> line and the <b>To</b> , <b>Cc</b> , and <b>Bcc</b> lists. Other implementation-defined headers may also be presented for editing. If the field is written with an initial value, it can be edited as if it had just been typed.                                                                                         |
| 23390 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23391 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23392 | <b>~i</b> <i>string</i>                                                                                     | Insert the value of the named variable, followed by a <newline>, into the text of the message. If the string is unset or null, the message shall not be changed.                                                                                                                                                                                                                      |
| 23393 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23394 | <b>~m</b> [ <i>msglist</i> ]                                                                                | Insert the specified messages into the message, prefixing non-empty lines with the string in the <b>indentprefix</b> variable. This command escape also shall insert message headers into the message, with field selection affected by the <b>discard</b> , <b>ignore</b> , and <b>retain</b> commands.                                                                              |
| 23395 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23396 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23397 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23398 | <b>~M</b> [ <i>msglist</i> ]                                                                                | This shall be the equivalent of the <b>~m</b> command escape, except that all headers shall be included in the message, regardless of previous <b>discard</b> , <b>ignore</b> , and <b>retain</b> commands.                                                                                                                                                                           |
| 23399 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23400 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23401 | <b>~p</b>                                                                                                   | Write the message being entered. If the message is longer than <b>crt</b> lines (see <b>Internal Variables in mailx</b> (on page 593)), the output shall be paginated as described for the <i>PAGER</i> variable.                                                                                                                                                                     |
| 23402 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23403 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23404 | <b>~q</b>                                                                                                   | Quit (see the <b>quit</b> command) from input mode by simulating an interrupt. If the body of the message is not empty, the partial message shall be saved in the dead-letter file. See <i>DEAD</i> for a description of this file.                                                                                                                                                   |
| 23405 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23406 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23407 | <b>~r</b> <i>file</i> , <b>~&lt;</b> <i>file</i> , <b>~r</b> <i>!command</i> , <b>~&lt;</b> <i>!command</i> |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23408 |                                                                                                             | Read in the file specified by the pathname <i>file</i> . If the argument begins with an exclamation mark ('!'), the rest of the string shall be taken as an arbitrary system command; the command interpreter specified by <i>SHELL</i> shall be invoked with two arguments: <b>-c</b> and <i>command</i> . The standard output of <i>command</i> shall be inserted into the message. |
| 23409 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23410 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23411 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23412 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23413 | <b>~s</b> <i>string</i>                                                                                     | Set the subject line to <i>string</i> .                                                                                                                                                                                                                                                                                                                                               |
| 23414 | <b>~t</b> <i>name...</i>                                                                                    | Add the given <i>names</i> to the <b>To</b> list.                                                                                                                                                                                                                                                                                                                                     |
| 23415 | <b>~v</b>                                                                                                   | Invoke the full-screen editor, as specified by the <i>VISUAL</i> environment variable, on the partial message.                                                                                                                                                                                                                                                                        |
| 23416 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23417 | <b>~w</b> <i>file</i>                                                                                       | Write the partial message, without the header, onto the file named by the pathname <i>file</i> . The file shall be created or the message shall be appended to it if the file exists.                                                                                                                                                                                                 |
| 23418 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |
| 23419 |                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                       |

23420       ~x           Exit as with ~q, except the message shall not be saved in the dead-letter file.

23421       ~| *command* Pipe the body of the message through the given *command* by invoking the  
 23422           command interpreter specified by *SHELL* with two arguments: -c and *command*.  
 23423           If the *command* returns a successful exit status, the standard output of the  
 23424           command shall replace the message. Otherwise, the message shall remain  
 23425           unchanged. If the *command* fails, an error message giving the exit status shall be  
 23426           written.

#### 23427 **EXIT STATUS**

23428       When the -e option is specified, the following exit values are returned:

23429       0   Mail was found.

23430       >0   Mail was not found or an error occurred.

23431       Otherwise, the following exit values are returned:

23432       0   Successful completion; note that this status implies that all messages were *sent*, but it gives  
 23433           no assurances that any of them were actually *delivered*.

23434       >0   An error occurred.

#### 23435 **CONSEQUENCES OF ERRORS**

23436       When in input mode (Receive Mode) or Send Mode:

23437       • If an error is encountered processing a command escape (see **Command Escapes in mailx**  
 23438         (on page 604)), a diagnostic message shall be written to standard error, and the message  
 23439         being composed may be modified, but this condition shall not prevent the message from  
 23440         being sent.

23441       • Other errors shall prevent the sending of the message.

23442       When in command mode:

23443       • Default.

#### 23444 **APPLICATION USAGE**

23445       Delivery of messages to remote systems requires the existence of communication paths to such  
 23446       systems. These need not exist.

23447       Input lines are limited to {LINE\_MAX} bytes, but mailers between systems may impose more  
 23448       severe line-length restrictions. This volume of IEEE Std 1003.1-2001 does not place any  
 23449       restrictions on the length of messages handled by *mailx*, and for delivery of local messages the  
 23450       only limitations should be the normal problems of available disk space for the target mail file.  
 23451       When sending messages to external machines, applications are advised to limit messages to less  
 23452       than 100 000 bytes because some mail gateways impose message-length restrictions.

23453       The format of the system mailbox is intentionally unspecified. Not all systems implement  
 23454       system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some  
 23455       system mailboxes may be multiple files, others records in a database. The internal format of the  
 23456       messages themselves is specified with the historical format from Version 7, but only after the  
 23457       messages have been saved in some file other than the system mailbox. This was done so that  
 23458       many historical applications expecting text-file mailboxes are not broken.

23459       Some new formats for messages can be expected in the future, probably including binary data,  
 23460       bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from  
 23461       handling such messages, but it must store them as text files in secondary mailboxes (unless  
 23462       some extension, such as a variable or command line option, is used to change the stored format).  
 23463       Its method of doing so is implementation-defined and might include translating the data into

23464 text file-compatible or readable form or omitting certain portions of the message from the stored  
23465 output.

23466 The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain**  
23467 command discards all header-fields except those explicitly retained. The **discard** command  
23468 keeps all header-fields except those explicitly discarded. If headers exist on the retained header  
23469 list, **discard** and **ignore** commands are ignored.

#### 23470 EXAMPLES

23471 None.

#### 23472 RATIONALE

23473 The standard developers felt strongly that a method for applications to send messages to  
23474 specific users was necessary. The obvious example is a batch utility, running non-interactively,  
23475 that wishes to communicate errors or results to a user. However, the actual format, delivery  
23476 mechanism, and method of reading the message are clearly beyond the scope of this volume of  
23477 IEEE Std 1003.1-2001.

23478 The intent of this command is to provide a simple, portable interface for sending messages non-  
23479 interactively. It merely defines a “front-end” to the historical mail system. It is suggested that  
23480 implementations explicitly denote the sender and recipient in the body of the delivered message.  
23481 Further specification of formats for either the message envelope or the message itself were  
23482 deliberately not made, as the industry is in the midst of changing from the current standards to a  
23483 more internationalized standard and it is probably incorrect, at this time, to require either one.

23484 Implementations are encouraged to conform to the various delivery mechanisms described in  
23485 the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request  
23486 for Comment (RFC) documents RFC 819, RFC 822, RFC 920, RFC 921, and RFC 1123.

23487 Many historical systems modified each body line that started with **From** by prefixing the ‘F’  
23488 with ‘>’. It is unnecessary, but allowed, to do that when the string does not follow a blank line  
23489 because it cannot be confused with the next header.

23490 The **edit** and **visual** commands merely edit the specified messages in a temporary file. They do  
23491 not modify the contents of those messages in the mailbox; such a capability could be added as an  
23492 extension, such as by using different command names.

23493 The restriction on a subject line being {LINE\_MAX}–10 bytes is based on the historical format  
23494 that consumes 10 bytes for **Subject:** and the trailing <newline>. Many historical mailers that a  
23495 message may encounter on other systems are not able to handle lines that long, however.

23496 Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient  
23497 justification to exclude this utility from this volume of IEEE Std 1003.1-2001. It is also arguable  
23498 that it is, in fact, testable, but that the tests themselves are not portable.

23499 When *mailx* is being used by an application that wishes to receive the results as if none of the  
23500 User Portability Utilities option features were supported, the *DEAD* environment variable must  
23501 be set to **/dev/null**. Otherwise, it may be subject to the file creations described in *mailx*  
23502 ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to  
23503 **/dev/null**, historical versions of *mailx* and *Mail* read initialization commands from a file before  
23504 processing begins. Since the initialization that a user specifies could alter the contents of  
23505 messages an application is trying to send, such applications must set *MAILRC* to **/dev/null**.

23506 The description of *LC\_TIME* uses “may affect” because many historical implementations do not  
23507 or cannot manipulate the date and time strings in the incoming mail headers. Some headers  
23508 found in incoming mail do not have enough information to determine the timezone in which the  
23509 mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal  
23510 form that then is parsed by routines like *strftime()* that can take *LC\_TIME* settings into account.

23511 Changing all these times to a user-specified format is allowed, but not required.

23512 The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System  
23513 V historical practice of using *pg* as the default. Bypassing the pagination function, such as by  
23514 declaring that *cat* is the paginator, would not meet with the intended meaning of this  
23515 description. However, any “portable user” would have to set *PAGER* explicitly to get his or her  
23516 preferred paginator on all systems. The paginator choice was made partially unspecified, unlike  
23517 the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common  
23518 theme of user input, whereas editors differ dramatically.

23519 Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to  
23520 be format details and were omitted.

23521 A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them  
23522 were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an  
23523 appropriate marketing distinction between systems.

23524 In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file*  
23525 option-argument to **-f**. By making *file* an operand, the guidelines are satisfied and users remain  
23526 portable. However, it does force implementations to support usage such as:

23527 `mailx -fin mymail.box`

23528 The **no name** method of unsetting variables is not present in all historical systems, but it is in  
23529 System V and provides a logical set of commands corresponding to the format of the display of  
23530 options from the *mailx set* command without arguments.

23531 The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the  
23532 same feature. They are synonyms in this volume of IEEE Std 1003.1-2001.

23533 The *mailx echo* command was not documented in the BSD version and has been omitted here  
23534 because it is not obviously useful for interactive users.

23535 The default prompt on the System V *mailx* is a question mark, on BSD *Mail* an ampersand. Since  
23536 this volume of IEEE Std 1003.1-2001 chose the *mailx* name, it kept the System V default,  
23537 assuming that BSD users would not have difficulty with this minor incompatibility (that they  
23538 can override).

23539 The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again,  
23540 since this volume of IEEE Std 1003.1-2001 chose the *mailx* name, it kept the System V default, but  
23541 allows the SunOS user to achieve the desired results using **flipr**, an internal variable in System V  
23542 *mailx*, although it has not been documented in the SVID.

23543 The **indentprefix** variable, the **retain** and **unalias** commands, and the **~F** and **~M** command  
23544 escapes were adopted from 4.3 BSD *Mail*.

23545 The **version** command was not included because no sufficiently general specification of the  
23546 version information could be devised that would still be useful to a portable user. This  
23547 command name should be used by suppliers who wish to provide version information about the  
23548 *mailx* command.

23549 The “implementation-specific (unspecified) system start-up file” historically has been named  
23550 **/etc/mailx.rc**, but this specific name and location are not required.

23551 The intent of the wording for the **next** command is that if any command has already displayed  
23552 the current message it should display a following message, but, otherwise, it should display the  
23553 current message. Consider the command sequence:

23554 `next 3`  
23555 `delete 3`

|       |                                                                                                                                                 |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| 23556 | next                                                                                                                                            |
| 23557 | where the <b>autoprint</b> option was not set. The normative text specifies that the second <b>next</b>                                         |
| 23558 | command should display a message following the third message, because even though the                                                           |
| 23559 | current message has not been displayed since it was set by the <b>delete</b> command, it has been                                               |
| 23560 | displayed since the current message was anything other than message number 3. This does not                                                     |
| 23561 | always match historical practice in some implementations, where the command file address                                                        |
| 23562 | followed by <b>next</b> (or the default command) would skip the message for which the user had                                                  |
| 23563 | searched.                                                                                                                                       |
| 23564 | <b>FUTURE DIRECTIONS</b>                                                                                                                        |
| 23565 | None.                                                                                                                                           |
| 23566 | <b>SEE ALSO</b>                                                                                                                                 |
| 23567 | Chapter 2 (on page 29), <i>ed</i> , <i>ls</i> , <i>more</i> , <i>vi</i>                                                                         |
| 23568 | <b>CHANGE HISTORY</b>                                                                                                                           |
| 23569 | First released in Issue 2.                                                                                                                      |
| 23570 | <b>Issue 5</b>                                                                                                                                  |
| 23571 | The description of the <i>EDITOR</i> environment variable is changed to indicate that <i>ed</i> is the default                                  |
| 23572 | editor if this variable is not set. In previous issues, this default was not stated explicitly at this                                          |
| 23573 | point but was implied further down in the text.                                                                                                 |
| 23574 | The FUTURE DIRECTIONS section is added.                                                                                                         |
| 23575 | <b>Issue 6</b>                                                                                                                                  |
| 23576 | The following new requirements on POSIX implementations derive from alignment with the                                                          |
| 23577 | Single UNIX Specification:                                                                                                                      |
| 23578 | <ul style="list-style-type: none"> <li>• The <b>-F</b> option is added.</li> </ul>                                                              |
| 23579 | <ul style="list-style-type: none"> <li>• The <b>allnet</b>, <b>debug</b>, and <b>sendwait</b> internal variables are added.</li> </ul>          |
| 23580 | <ul style="list-style-type: none"> <li>• The <b>C</b>, <b>ec</b>, <b>fo</b>, <b>F</b>, and <b>S</b> <i>mailx</i> commands are added.</li> </ul> |
| 23581 | In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating “ <i>HOME</i>                                                               |
| 23582 | directory” is replaced by “directory referred to by the <i>HOME</i> environment variable”.                                                      |
| 23583 | The <i>mailx</i> utility is aligned with the IEEE P1003.2b draft standard, which includes various                                               |
| 23584 | clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2:1993                                                          |
| 23585 | standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11,                                                    |
| 23586 | #103, #106, #108, #114, #115, #122, and #129.                                                                                                   |
| 23587 | The normative text is reworded to avoid use of the term “must” for application requirements.                                                    |
| 23588 | The <i>TZ</i> entry is added to the ENVIRONMENT VARIABLES section.                                                                              |
| 23589 | IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/32 is applied, applying a change to the                                                        |
| 23590 | EXTENDED DESCRIPTION, raised by IEEE PASC Interpretation 1003.2 #122, which was                                                                 |
| 23591 | overlooked in the first version of IEEE Std 1003.1-2001.                                                                                        |

23592 **NAME**23593       make — maintain, update, and regenerate groups of programs (**DEVELOPMENT**)23594 **SYNOPSIS**

```
23595 sd make [-einpqrst] [-f makefile] ... [-k | -S] [macro=value] ...
23596 [target_name...]
```

23597

23598 **DESCRIPTION**

23599       The *make* utility shall update files that are derived from other files. A typical case is one where  
 23600       object files are derived from the corresponding source files. The *make* utility examines time  
 23601       relationships and shall update those derived files (called targets) that have modified times  
 23602       earlier than the modified times of the files (called prerequisites) from which they are derived. A  
 23603       description file (makefile) contains a description of the relationships between files, and the  
 23604       commands that need to be executed to update the targets to reflect changes in their  
 23605       prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and  
 23606       optional commands to be executed when a prerequisite is newer than the target. There are two  
 23607       types of rule:

23608           1. *Inference rules*, which have one target name with at least one period ( '.' ) and no slash  
 23609           (' / ')

23610           2. *Target rules*, which can have more than one target name

23611       In addition, *make* shall have a collection of built-in macros and inference rules that infer  
 23612       prerequisite relationships to simplify maintenance of programs.

23613       To receive exactly the behavior described in this section, the user shall ensure that a portable  
 23614       makefile shall:

- 23615       • Include the special target **.POSIX**
- 23616       • Omit any special target reserved for implementations (a leading period followed by  
 23617       uppercase letters) that has not been specified by this section

23618       The behavior of *make* is unspecified if either or both of these conditions are not met.

23619 **OPTIONS**

23620       The *make* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 23621       12.2, Utility Syntax Guidelines.

23622       The following options shall be supported:

23623       **-e**           Cause environment variables, including those with null values, to override macro  
 23624       assignments within makefiles.

23625       **-f *makefile***   Specify a different makefile. The argument *makefile* is a pathname of a description  
 23626       file, which is also referred to as the *makefile*. A pathname of '-' shall denote the  
 23627       standard input. There can be multiple instances of this option, and they shall be  
 23628       processed in the order specified. The effect of specifying the same option-  
 23629       argument more than once is unspecified.

23630       **-i**           Ignore error codes returned by invoked commands. This mode is the same as if the  
 23631       special target **.IGNORE** were specified without prerequisites.

23632       **-k**           Continue to update other targets that do not depend on the current target if a non-  
 23633       ignored error occurs while executing the commands to bring a target up-to-date.

23634       **-n**           Write commands that would be executed on standard output, but do not execute  
 23635       them. However, lines with a plus sign ( '+' ) prefix shall be executed. In this mode,

lines with an at sign ('@') character prefix shall be written to standard output.

**-p** Write to standard output the complete set of macro definitions and target descriptions. The output format is unspecified.

**-q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit value of 1. Targets shall not be updated if this option is specified. However, a makefile command line (associated with the targets) with a plus sign ('+') prefix shall be executed.

**-r** Clear the suffix list and do not use the built-in rules.

**-S** Terminate *make* if an error occurs while executing the commands to bring a target up-to-date. This shall be the default and the opposite of **-k**.

**-s** Do not write makefile command lines or touch messages (see **-t**) to standard output before executing. This mode shall be the same as if the special target **.SILENT** were specified without prerequisites.

**-t** Update the modification time of each target as though a *touch target* had been executed. Targets that have prerequisites but no commands (see **Target Rules** (on page 614)), or that are already up-to-date, shall not be touched in this manner. Write messages to standard output for each target file indicating the name of the file and that it was touched. Normally, the *makefile* command lines associated with each target are not executed. However, a command line with a plus sign ('+') prefix shall be executed.

Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any options specified on the *make* utility command line. If the **-k** and **-S** options are both specified on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option specified shall take precedence. If the **-f** or **-p** options appear in the *MAKEFLAGS* environment variable, the result is undefined.

## 23661 OPERANDS

The following operands shall be supported:

*target\_name* Target names, as defined in the EXTENDED DESCRIPTION section. If no target is specified, while *make* is processing the makefiles, the first target that *make* encounters that is not a special target or an inference rule shall be used.

*macro=value* Macro definitions, as defined in **Macros** (on page 616).

If the *target\_name* and *macro=value* operands are intermixed on the *make* utility command line, the results are unspecified.

## 23669 STDIN

The standard input shall be used only if the *makefile* option-argument is '-'. See the INPUT FILES section.

## 23672 INPUT FILES

The input file, otherwise known as the makefile, is a text file containing rules, macro definitions, and comments. See the EXTENDED DESCRIPTION section.

## 23675 ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *make*:

*LANG* Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

|       |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 23681 | <i>LC_ALL</i>              | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 23682 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23683 | <i>LC_CTYPE</i>            | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 23684 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23685 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23686 | <i>LC_MESSAGES</i>         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23687 |                            | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 23688 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23689 | <i>MAKEFLAGS</i>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23690 |                            | This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):                                                                                                                                                                                                                                                                                                                                                                        |
| 23691 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23692 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23693 |                            | • The characters are option letters without the leading hyphens or <blank> separation used on a <i>make</i> utility command line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 23694 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23695 |                            | • The characters are formatted in a manner similar to a portion of the <i>make</i> utility command line: options are preceded by hyphens and <blank>-separated as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines. The <i>macro=value</i> macro definition operands can also be included. The difference between the contents of <i>MAKEFLAGS</i> and the <i>make</i> utility command line is that the contents of the variable shall not be subjected to the word expansions (see Section 2.6 (on page 36)) associated with parsing the command line values. |
| 23696 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23697 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23698 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23699 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23700 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23701 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23702 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23703 | XSI <i>NLSPATH</i>         | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 23704 | XSI <i>PROJECTDIR</i>      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23705 |                            | Provide a directory to be used to search for SCCS files not found in the current directory. In all of the following cases, the search for SCCS files is made in the directory <b>SCCS</b> in the identified directory. If the value of <i>PROJECTDIR</i> begins with a slash, it shall be considered an absolute pathname; otherwise, the value of <i>PROJECTDIR</i> is treated as a user name and that user's initial working directory shall be examined for a subdirectory <b>src</b> or <b>source</b> . If such a directory is found, it shall be used. Otherwise, the value is used as a relative pathname.        |
| 23706 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23707 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23708 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23709 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23710 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23711 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23712 |                            | If <i>PROJECTDIR</i> is not set or has a null value, the search for SCCS files shall be made in the directory <b>SCCS</b> in the current directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 23713 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23714 |                            | The setting of <i>PROJECTDIR</i> affects all files listed in the remainder of this utility description for files with a component named <b>SCCS</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 23715 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23716 |                            | The value of the <i>SHELL</i> environment variable shall not be used as a macro and shall not be modified by defining the <b>SHELL</b> macro in a makefile or on the command line. All other environment variables, including those with null values, shall be used as macros, as defined in <b>Macros</b> (on page 616).                                                                                                                                                                                                                                                                                               |
| 23717 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23718 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23719 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23720 | <b>ASYNCHRONOUS EVENTS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23721 |                            | If not already ignored, <i>make</i> shall trap SIGHUP, SIGTERM, SIGINT, and SIGQUIT and remove the current target unless the target is a directory or the target is a prerequisite of the special target <b>.PRECIOUS</b> or unless one of the <b>-n</b> , <b>-p</b> , or <b>-q</b> options was specified. Any targets removed in this manner shall be reported in diagnostic messages of unspecified format, written to standard error. After this cleanup process, if any, <i>make</i> shall take the standard action for all other signals.                                                                          |
| 23722 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23723 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23724 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23725 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 23726 |                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |



23727 **STDOUT**

23728 The *make* utility shall write all commands to be executed to standard output unless the **-s** option  
 23729 was specified, the command is prefixed with an at sign, or the special target **.SILENT** has either  
 23730 the current target as a prerequisite or has no prerequisites. If *make* is invoked without any work  
 23731 needing to be done, it shall write a message to standard output indicating that no action was  
 23732 taken. If the **-t** option is present and a file is touched, *make* shall write to standard output a  
 23733 message of unspecified format indicating that the file was touched, including the filename of the  
 23734 file.

23735 **STDERR**

23736 The standard error shall be used only for diagnostic messages.

23737 **OUTPUT FILES**

23738 Files can be created when the **-t** option is present. Additional files can also be created by the  
 23739 utilities invoked by *make*.

23740 **EXTENDED DESCRIPTION**

23741 The *make* utility attempts to perform the actions required to ensure that the specified targets are  
 23742 up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it  
 23743 does not exist. The *make* utility shall treat all prerequisites as targets themselves and recursively  
 23744 ensure that they are up-to-date, processing them in the order in which they appear in the rule.  
 23745 The *make* utility shall use the modification times of files to determine whether the corresponding  
 23746 targets are out-of-date.

23747 After *make* has ensured that all of the prerequisites of a target are up-to-date and if the target is  
 23748 out-of-date, the commands associated with the target entry shall be executed. If there are no  
 23749 commands listed for the target, the target shall be treated as up-to-date.

23750 **Makefile Syntax**

23751 A makefile can contain rules, macro definitions (see **Macros** (on page 616)), and comments.  
 23752 There are two kinds of rules: *inference rules* and *target rules*. The *make* utility shall contain a set of  
 23753 built-in inference rules. If the **-r** option is present, the built-in rules shall not be used and the  
 23754 suffix list shall be cleared. Additional rules of both types can be specified in a makefile. If a rule  
 23755 is defined more than once, the value of the rule shall be that of the last one specified. Macros can  
 23756 also be defined more than once, and the value of the macro is specified in **Macros** (on page 616).  
 23757 Comments start with a number sign ('#') and continue until an unescaped <newline> is  
 23758 reached.

23759 By default, the following files shall be tried in sequence: **./makefile** and **./Makefile**. If neither  
 23760 XSI **./makefile** or **./Makefile** are found, other implementation-defined files may also be tried. On  
 23761 XSI-conformant systems, the additional files **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and  
 23762 **SCCS/s.Makefile** shall also be tried.

23763 The **-f** option shall direct *make* to ignore any of these default files and use the specified argument  
 23764 as a makefile instead. If the **-** argument is specified, standard input shall be used.

23765 The term *makefile* is used to refer to any rules provided by the user, whether in **./makefile** or its  
 23766 variants, or specified by the **-f** option.

23767 The rules in makefiles shall consist of the following types of lines: target rules, including special  
 23768 targets (see **Target Rules** (on page 614)), inference rules (see **Inference Rules** (on page 617)),  
 23769 macro definitions (see **Macros** (on page 616)), empty lines, and comments.

23770 When an escaped <newline> (one preceded by a backslash) is found anywhere in the makefile  
 23771 except in a command line, it shall be replaced, along with any leading white space on the  
 23772 following line, with a single <space>. When an escaped <newline> is found in a command line

23773 in a makefile, the command line shall contain the backslash, the <newline>, and the next line,  
 23774 except that the first character of the next line shall not be included if it is a <tab>.

## 23775 **Makefile Execution**

23776 Makefile command lines shall be processed one at a time by writing the makefile command line  
 23777 to the standard output (unless one of the conditions listed under '@' suppresses the writing)  
 23778 and executing the command(s) in the line. A <tab> may precede the command to standard  
 23779 output. Command execution shall be as if the makefile command line were the argument to the  
 23780 *system()* function. The environment for the command being executed shall contain all of the  
 23781 variables in the environment of *make*.

23782 By default, when *make* receives a non-zero status from the execution of a command, it shall  
 23783 terminate with an error message to standard error.

23784 Makefile command lines can have one or more of the following prefixes: a hyphen ('-'), an at  
 23785 sign ('@'), or a plus sign ('+'). These shall modify the way in which *make* processes the  
 23786 command. When a command is written to standard output, the prefix shall not be included in  
 23787 the output.

23788 – If the command prefix contains a hyphen, or the **-i** option is present, or the special target  
 23789 **.IGNORE** has either the current target as a prerequisite or has no prerequisites, any error  
 23790 found while executing the command shall be ignored.

23791 @ If the command prefix contains an at sign and the *make* utility command line **-n** option is  
 23792 not specified, or the **-s** option is present, or the special target **.SILENT** has either the current  
 23793 target as a prerequisite or has no prerequisites, the command shall not be written to  
 23794 standard output before it is executed.

23795 + If the command prefix contains a plus sign, this indicates a makefile command line that  
 23796 shall be executed even if **-n**, **-q**, or **-t** is specified.

## 23797 **Target Rules**

23798 Target rules are formatted as follows:

```
23799 target [target...]: [prerequisite...] [;command]
23800 [

```

23803 *line that does not begin with <tab>*

23804 Target entries are specified by a <blank>-separated, non-null list of targets, then a colon, then a  
 23805 <blank>-separated, possibly empty list of prerequisites. Text following a semicolon, if any, and  
 23806 all following lines that begin with a <tab>, are makefile command lines to be executed to update  
 23807 the target. The first non-empty line that does not begin with a <tab> or '#' shall begin a new  
 23808 entry. An empty or blank line, or a line beginning with '#', may begin a new entry.

23809 Applications shall select target names from the set of characters consisting solely of periods,  
 23810 underscores, digits, and alphabetic characters from the portable character set (see the Base Definitions  
 23811 volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set). Implementations may allow  
 23812 other characters in target names as extensions. The interpretation of targets containing the  
 23813 characters '%' and '"' is implementation-defined.

23814 A target that has prerequisites, but does not have any commands, can be used to add to the  
 23815 prerequisite list for that target. Only one target rule for any given target can contain commands.

23816 Lines that begin with one of the following are called *special targets* and control the operation of  
 23817 *make*:

23818 **.DEFAULT** If the makefile uses this special target, the application shall ensure that it is  
 23819 specified with commands, but without prerequisites. The commands shall be used  
 23820 by *make* if there are no other rules available to build a target.

23821 **.IGNORE** Prerequisites of this special target are targets themselves; this shall cause errors  
 23822 from commands associated with them to be ignored in the same manner as  
 23823 specified by the **-i** option. Subsequent occurrences of **.IGNORE** shall add to the  
 23824 list of targets ignoring command errors. If no prerequisites are specified, *make* shall  
 23825 behave as if the **-i** option had been specified and errors from all commands  
 23826 associated with all targets shall be ignored.

23827 **.POSIX** The application shall ensure that this special target is specified without  
 23828 prerequisites or commands. If it appears as the first non-comment line in the  
 23829 makefile, *make* shall process the makefile as specified by this section; otherwise, the  
 23830 behavior of *make* is unspecified.

23831 **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the  
 23832 asynchronous events explicitly described in the ASYNCHRONOUS EVENTS  
 23833 section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious  
 23834 files. If no prerequisites are specified, all targets in the makefile shall be treated as  
 23835 if specified with **.PRECIOUS**.

23836 XSI **.SCCS\_GET** The application shall ensure that this special target is specified without  
 23837 prerequisites. If this special target is included in a makefile, the commands  
 23838 specified with this target shall replace the default commands associated with this  
 23839 special target (see **Default Rules** (on page 620)). The commands specified with  
 23840 this target are used to get all SCCS files that are not found in the current directory.

23841 When source files are named in a dependency list, *make* shall treat them just like  
 23842 any other target. Because the source file is presumed to be present in the directory,  
 23843 there is no need to add an entry for it to the makefile. When a target has no  
 23844 dependencies, but is present in the directory, *make* shall assume that that file is up-  
 23845 to-date. If, however, an SCCS file named **SCCS/s.source\_file** is found for a target  
 23846 *source\_file*, *make* compares the timestamp of the target file with that of the  
 23847 **SCCS/s.source\_file** to ensure the target is up-to-date. If the target is missing, or if  
 23848 the SCCS file is newer, *make* shall automatically issue the commands specified for  
 23849 the **.SCCS\_GET** special target to retrieve the most recent version. However, if the  
 23850 target is writable by anyone, *make* shall not retrieve a new version.

23851 **.SILENT** Prerequisites of this special target are targets themselves; this shall cause  
 23852 commands associated with them not to be written to the standard output before  
 23853 they are executed. Subsequent occurrences of **.SILENT** shall add to the list of  
 23854 targets with silent commands. If no prerequisites are specified, *make* shall behave  
 23855 as if the **-s** option had been specified and no commands or touch messages  
 23856 associated with any target shall be written to standard output.

23857 **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are  
 23858 used in conjunction with the inference rules (see **Inference Rules** (on page 617)). If  
 23859 **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be  
 23860 cleared.

23861 The special targets **.IGNORE**, **.POSIX**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES** shall be specified  
 23862 without commands.

23863 Targets with names consisting of a leading period followed by the uppercase letters "POSIX"  
 23864 and then any other characters are reserved for future standardization. Targets with names  
 23865 consisting of a leading period followed by one or more uppercase letters are reserved for  
 23866 implementation extensions.

## 23867 **Macros**

23868 Macro definitions are in the form:

23869 *string1* = [*string2*]

23870 The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all  
 23871 characters, if any, after the equal sign, up to a comment character ('#') or an unescaped  
 23872 <newline>. Any <blank>s immediately before or after the equal sign shall be ignored.

23873 Applications shall select macro names from the set of characters consisting solely of periods,  
 23874 underscores, digits, and alphabets from the portable character set (see the Base Definitions  
 23875 volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set). A macro name shall not  
 23876 contain an equals sign. Implementations may allow other characters in macro names as  
 23877 extensions.

23878 Macros can appear anywhere in the makefile. Macro expansions using the forms  $\$(string1)$  or  
 23879  $\${string1}$  shall be replaced by *string2*, as follows:

- 23880 • Macros in target lines shall be evaluated when the target line is read.
- 23881 • Macros in makefile command lines shall be evaluated when the command is executed.
- 23882 • Macros in the string before the equals sign in a macro definition shall be evaluated when the  
 23883 macro assignment is made.
- 23884 • Macros after the equals sign in a macro definition shall not be evaluated until the defined  
 23885 macro is used in a rule or command, or before the equals sign in a macro definition.

23886 The parentheses or braces are optional if *string1* is a single character. The macro  $\$\$$  shall be  
 23887 replaced by the single character '\$'. If *string1* in a macro expansion contains a macro  
 23888 expansion, the results are unspecified.

23889 Macro expansions using the forms  $\$(string1[:subst1=[subst2]])$  or  $\${string1[:subst1=[subst2]]}$  can  
 23890 be used to replace all occurrences of *subst1* with *subst2* when the macro substitution is  
 23891 performed. The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word  
 23892 in *string1* (where a *word*, in this context, is defined to be a string delimited by the beginning of  
 23893 the line, a <blank>, or a <newline>). If *string1* in a macro expansion contains a macro expansion,  
 23894 the results are unspecified.

23895 Macro expansions in *string1* of macro definition lines shall be evaluated when read. Macro  
 23896 expansions in *string2* of macro definition lines shall be performed when the macro identified by  
 23897 *string1* is expanded in a rule or command.

23898 Macro definitions shall be taken from the following sources, in the following logical order,  
 23899 before the makefile(s) are read.

- 23900 1. Macros specified on the *make* utility command line, in the order specified on the command  
 23901 line. It is unspecified whether the internal macros defined in **Internal Macros** (on page 619)  
 23902 are accepted from this source.
- 23903 2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the  
 23904 environment variable. It is unspecified whether the internal macros defined in **Internal**  
 23905 **Macros** (on page 619) are accepted from this source.

3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and including the variables with null values.
4. Macros defined in the inference rules built into *make*.

Macro definitions from these sources shall not override macro definitions from a lower-numbered source. Macro definitions from a single source (for example, the *make* utility command line, the *MAKEFLAGS* environment variable, or the other environment variables) shall override previous macro definitions from the same source.

Macros defined in the makefile(s) shall override macro definitions that occur before them in the makefile(s) and macro definitions from source 4. If the *-e* option is not specified, macros defined in the makefile(s) shall override macro definitions from source 3. Macros defined in the makefile(s) shall not override macro definitions from source 1 or source 2.

Before the makefile(s) are read, all of the *make* utility command line options (except *-f* and *-p*) and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance of the *make* command, the original macro's value is recovered. Other implementation-defined options and macros may also be added to the *MAKEFLAGS* macro. If this modifies the value of the *MAKEFLAGS* macro, or, if the *MAKEFLAGS* macro is modified at any subsequent time, the *MAKEFLAGS* environment variable shall be modified to match the new value of the *MAKEFLAGS* macro. The result of setting *MAKEFLAGS* in the Makefile is unspecified.

Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other implementation-defined variables may also be added to the environment of *make*.

The **SHELL** macro shall be treated specially. It shall be provided by *make* and set to the pathname of the shell command language interpreter (see *sh*). The *SHELL* environment variable shall not affect the value of the **SHELL** macro. If **SHELL** is defined in the makefile or is specified on the command line, it shall replace the original value of the **SHELL** macro, but shall not affect the *SHELL* environment variable. Other effects of defining **SHELL** in the makefile or on the command line are implementation-defined.

## Inference Rules

Inference rules are formatted as follows:

```
target:
<tab>command
[<tab>command]
...
line that does not begin with <tab> or #
```

The application shall ensure that the *target* portion is a valid target name (see **Target Rules** (on page 614)) of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as prerequisites of the **.SUFFIXES** special target and *s1* and *s2* do not contain any slashes or periods.) If there is only one period in the target, it is a single-suffix inference rule. Targets with two periods are double-suffix inference rules. Inference rules can have only one target before the colon.

The application shall ensure that the makefile does not specify prerequisites for inference rules; no characters other than white space shall follow the colon in the first line, except when creating the *empty rule*, described below. Prerequisites are inferred, as described below.

23951 Inference rules can be redefined. A target that matches an existing inference rule shall overwrite  
 23952 the old inference rule. An empty rule can be created with a command consisting of simply a  
 23953 semicolon (that is, the rule still exists and is found during inference rule search, but since it is  
 23954 empty, execution has no effect). The empty rule can also be formatted as follows:

23955 `rule: ;`

23956 where zero or more <blank>s separate the colon and semicolon.

23957 The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be  
 23958 made up-to-date. A list of inference rules defines the commands to be executed. By default, *make*  
 23959 contains a built-in set of inference rules. Additional rules can be specified in the makefile.

23960 The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by  
 23961 the inference rules. The order in which the suffixes are specified defines the order in which the  
 23962 inference rules for the suffixes are used. New suffixes shall be appended to the current list by  
 23963 specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites  
 23964 shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is  
 23965 required to change the order of the suffixes.

23966 Normally, the user would provide an inference rule for each suffix. The inference rule to update  
 23967 a target with a suffix **.s1** from a prerequisite with a suffix **.s2** is specified as a target **.s2.s1**. The  
 23968 internal macros provide the means to specify general inference rules (see **Internal Macros** (on  
 23969 page 619)).

23970 When no target rule is found to update a target, the inference rules shall be checked. The suffix  
 23971 of the target (**.s1**) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special  
 23972 targets. If the **.s1** suffix is found in **.SUFFIXES**, the inference rules shall be searched in the order  
 23973 defined for the first **.s2.s1** rule whose prerequisite file (**\$.s2**) exists. If the target is out-of-date  
 23974 with respect to this prerequisite, the commands for that inference rule shall be executed.

23975 If the target to be built does not contain a suffix and there is no rule for the target, the single  
 23976 suffix inference rules shall be checked. The single-suffix inference rules define how to build a  
 23977 target if a file is found with a name that matches the target name with one of the single suffixes  
 23978 appended. A rule with one suffix **.s2** is the definition of how to build *target* from **target.s2**. The  
 23979 other suffix (**.s1**) is treated as null.

23980 XSI A tilde ('~') in the above rules refers to an SCCS file in the current directory. Thus, the rule **.c~.o**  
 23981 would transform an SCCS C-language source file into an object file (**.o**). Because the **s.** of the  
 23982 SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the '~' is a way  
 23983 of changing any file reference into an SCCS file reference.

## 23984 Libraries

23985 If a target or prerequisite contains parentheses, it shall be treated as a member of an archive  
 23986 library. For the *lib(member.o)* expression *lib* refers to the name of the archive library and *member.o*  
 23987 to the member name. The application shall ensure that the member is an object file with the **.o**  
 23988 suffix. The modification time of the expression is the modification time for the member as kept  
 23989 in the archive library; see *ar*. The **.a** suffix shall refer to an archive library. The **.s2.a** rule shall be  
 23990 used to update a member in the library from a file with a suffix **.s2**.

23991      **Internal Macros**

23992      The *make* utility shall maintain five internal macros that can be used in target and inference rules.  
 23993      In order to clearly define the meaning of these macros, some clarification of the terms *target rule*,  
 23994      *inference rule*, *target*, and *prerequisite* is necessary.

23995      Target rules are specified by the user in a makefile for a particular target. Inference rules are  
 23996      user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites  
 23997      are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those  
 23998      prerequisites that are generated when inference rules are used. Inference rules are applied to  
 23999      implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in  
 24000      the makefile. Target rules are applied to targets specified in the makefile.

24001      Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit)  
 24002      shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon  
 24003      recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be  
 24004      processed recursively until a target is found that has no prerequisites, at which point the  
 24005      recursion stops. The recursion shall then back up, updating each target as it goes.

24006      In the definitions that follow, the word *target* refers to one of:

- 24007      • A target specified in the makefile
- 24008      • An explicit prerequisite specified in the makefile that becomes the target when *make*  
 24009      processes it during recursion
- 24010      • An implicit prerequisite that becomes a target when *make* processes it during recursion

24011      In the definitions that follow, the word *prerequisite* refers to one of the following:

- 24012      • An explicit prerequisite specified in the makefile for a particular target
- 24013      • An implicit prerequisite generated as a result of locating an appropriate inference rule and  
 24014      corresponding file that matches the suffix of the target

24015      The five internal macros are:

24016      **\$@**      The **\$@** shall evaluate to the full target name of the current target, or the archive  
 24017      filename part of a library archive target. It shall be evaluated for both target and  
 24018      inference rules.

24019      For example, in the **.c.a** inference rule, **\$@** represents the out-of-date **.a** file to be built.  
 24020      Similarly, in a makefile target rule to build **lib.a** from **file.c**, **\$@** represents the out-of-  
 24021      date **lib.a**.

24022      **\$%**      The **\$%** macro shall be evaluated only when the current target is an archive library  
 24023      member of the form *libname(member.o)*. In these cases, **\$@** shall evaluate to *libname* and  
 24024      **\$%** shall evaluate to *member.o*. The **\$%** macro shall be evaluated for both target and  
 24025      inference rules.

24026      For example, in a makefile target rule to build **lib.a(file.o)**, **\$%** represents **file.o**, as  
 24027      opposed to **\$@**, which represents **lib.a**.

24028      **\$?**      The **\$?** macro shall evaluate to the list of prerequisites that are newer than the current  
 24029      target. It shall be evaluated for both target and inference rules.

24030      For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and  
 24031      where *prog* is not out-of-date with respect to **file1.o**, but is out-of-date with respect to  
 24032      **file2.o** and **file3.o**, **\$?** represents **file2.o** and **file3.o**.

24033       \$<       In an inference rule, the \$< macro shall evaluate to the filename whose existence  
 24034       allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the \$<  
 24035       macro shall evaluate to the current target name. The meaning of the \$< macro shall be  
 24036       otherwise unspecified.

24037       For example, in the **.c.a** inference rule, \$< represents the prerequisite **.c** file.

24038       \$\*       The \$\* macro shall evaluate to the current target name with its suffix deleted. It shall be  
 24039       evaluated at least for inference rules.

24040       For example, in the **.c.a** inference rule, \$\*.o represents the out-of-date **.o** file that  
 24041       corresponds to the prerequisite **.c** file.

24042       Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended  
 24043       to any of the macros, the meaning shall be changed to the *directory part* for 'D' and *filename part*  
 24044       for 'F'. The directory part is the path prefix of the file without a trailing slash; for the current  
 24045       directory, the directory part is '.'. When the \$? macro contains more than one prerequisite  
 24046       filename, the \$(?D) and \$(?F) (or \${?D} and \${?F}) macros expand to a list of directory name parts  
 24047       and filename parts respectively.

24048       For the target *lib(member.o)* and the **s2.a** rule, the internal macros shall be defined as:

24049       \$<       *member.s2*

24050       \$\*       *member*

24051       \$@       *lib*

24052       \$?       *member.s2*

24053       \$%       *member.o*

## 24054       **Default Rules**

24055       The default rules for *make* shall achieve results that are the same as if the following were used.  
 24056       Implementations that do not support the C-Language Development Utilities option may omit  
 24057       **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l** inference rules.  
 24058       Implementations that do not support FORTRAN may omit **FC**, **FFLAGS**, and the **.f** inference  
 24059       rules. Implementations may provide additional macros and rules.

## 24060       *SPECIAL TARGETS*

24061 xSI       .SCCS\_GET: sccs \$(SCCSFLAGS) get \$(SCCSGETFLAGS) \$@

24062

24063 xSI       .SUFFIXES: .o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~

## 24064       **MACROS**

24065       MAKE=make

24066       AR=ar

24067       ARFLAGS=-rv

24068       YACC=yacc

24069       YFLAGS=

24070       LEX=lex

24071       LFLAGS=

24072       LDFLAGS=

24073       CC=c99

24074       CFLAGS=-O

24075       FC=fort77



```

24076 FFLAGS=-O 1
24077 xSI GET=get
24078 GFLAGS=
24079 SCCSFLAGS=
24080 SCCSGETFLAGS=-s
24081

```

#### 24082 *SINGLE SUFFIX RULES*

```

24083 .c:
24084 $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<
24085
24086 .f:
24087 $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $<
24088
24089 .sh:
24090 cp $< $@
24091 chmod a+x $@
24092

```

```

24090 xSI .c~:
24091 $(GET) $(GFLAGS) -p $< > $*.c
24092 $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
24093

```

```

24093 .f~:
24094 $(GET) $(GFLAGS) -p $< > $*.f
24095 $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f
24096

```

```

24096 .sh~:
24097 $(GET) $(GFLAGS) -p $< > $*.sh
24098 cp $*.sh $@
24099 chmod a+x $@
24100

```

#### 24101 *DOUBLE SUFFIX RULES*

```

24102 .c.o:
24103 $(CC) $(CFLAGS) -c $<
24104

```

```

24104 .f.o:
24105 $(FC) $(FFLAGS) -c $<
24106

```

```

24106 .y.o:
24107 $(YACC) $(YFLAGS) $<
24108 $(CC) $(CFLAGS) -c y.tab.c
24109 rm -f y.tab.c
24110 mv y.tab.o $@
24111

```

```

24111 .l.o:
24112 $(LEX) $(LFLAGS) $<
24113 $(CC) $(CFLAGS) -c lex.yy.c
24114 rm -f lex.yy.c
24115 mv lex.yy.o $@
24116

```

```

24116 .y.c:
24117 $(YACC) $(YFLAGS) $<
24118 mv y.tab.c $@
24119

```

```

24119 .l.c:
24120 $(LEX) $(LFLAGS) $<

```

```

24121 mv lex.yy.c $@
24122 xsi .c~.o:
24123 $(GET) $(GFLAGS) -p $< > $*.c
24124 $(CC) $(CFLAGS) -c $*.c
24125
24126 .f~.o:
24127 $(GET) $(GFLAGS) -p $< > $*.f
24128 $(FC) $(FFLAGS) -c $*.f
24129
24128 .y~.o:
24129 $(GET) $(GFLAGS) -p $< > $*.y
24130 $(YACC) $(YFLAGS) $*.y
24131 $(CC) $(CFLAGS) -c y.tab.c
24132 rm -f y.tab.c
24133 mv y.tab.o $@
24134
24134 .l~.o:
24135 $(GET) $(GFLAGS) -p $< > $*.l
24136 $(LEX) $(LFLAGS) $*.l
24137 $(CC) $(CFLAGS) -c lex.yy.c
24138 rm -f lex.yy.c
24139 mv lex.yy.o $@
24140
24140 .y~.c:
24141 $(GET) $(GFLAGS) -p $< > $*.y
24142 $(YACC) $(YFLAGS) $*.y
24143 mv y.tab.c $@
24144
24144 .l~.c:
24145 $(GET) $(GFLAGS) -p $< > $*.l
24146 $(LEX) $(LFLAGS) $*.l
24147 mv lex.yy.c $@
24148
24149 .c.a:
24150 $(CC) -c $(CFLAGS) $<
24151 $(AR) $(ARFLAGS) $@ $*.o
24152 rm -f $*.o
24153
24153 .f.a:
24154 $(FC) -c $(FFLAGS) $<
24155 $(AR) $(ARFLAGS) $@ $*.o
24156 rm -f $*.o
24157 EXIT STATUS
24158 When the -q option is specified, the make utility shall exit with one of the following values:
24159
24159 0 Successful completion.
24160
24160 1 The target was not up-to-date.
24161
24161 >1 An error occurred.
24162
24162 When the -q option is not specified, the make utility shall exit with one of the following values:
24163
24163 0 Successful completion.
24164
24164 >0 An error occurred.

```

24165 **CONSEQUENCES OF ERRORS**

24166 Default.

24167 **APPLICATION USAGE**

24168 If there is a source file (such as *./source.c*) and there are two SCCS files corresponding to it  
 24169 (*./s.source.c* and *./SCCS/s.source.c*), on XSI-conformant systems *make* uses the SCCS file in the  
 24170 current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*,  
 24171 *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for  
 24172 a given source file, future developers are very likely to be confused.

24173 It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to  
 24174 guarantee that they are not affected by local extensions.

24175 The **-k** and **-S** options are both present so that the relationship between the command line, the  
 24176 *MAKEFLAGS* variable, and the makefile can be controlled precisely. If the **k** flag is passed in  
 24177 *MAKEFLAGS* and a command is of the form:

24178 `$ (MAKE) -S foo`24179 then the default behavior is restored for the child *make*.

24180 When the **-n** option is specified, it is always added to *MAKEFLAGS*. This allows a recursive  
 24181 *make -n target* to be used to see all of the action that would be taken to update *target*.

24182 Because of widespread historical practice, interpreting a '#' number sign inside a variable as  
 24183 the start of a comment has the unfortunate side effect of making it impossible to place a number  
 24184 sign in a variable, thus forbidding something like:

24185 `CFLAGS = "-D COMMENT_CHAR='#'"`

24186 Many historical *make* utilities stop chaining together inference rules when an intermediate target  
 24187 is nonexistent. For example, it might be possible for a *make* to determine that both **.y.c** and **.c.o**  
 24188 could be used to convert a **.y** to a **.o**. Instead, in this case, *make* requires the use of a **.y.o** rule.

24189 The best way to provide portable makefiles is to include all of the rules needed in the makefile  
 24190 itself. The rules provided use only features provided by other parts of this volume of  
 24191 IEEE Std 1003.1-2001. The default rules include rules for optional commands in this volume of  
 24192 IEEE Std 1003.1-2001. Only rules pertaining to commands that are provided are needed in an  
 24193 implementation's default set.

24194 Macros used within other macros are evaluated when the new macro is used rather than when  
 24195 the new macro is defined. Therefore:

```
24196 MACRO = value1
24197 NEW = $(MACRO)
24198 MACRO = value2
```

```
24199 target:
24200 echo $(NEW)
```

24201 would produce *value2* and not *value1* since **NEW** was not expanded until it was needed in the  
 24202 *echo* command line.

24203 Some historical applications have been known to intermix *target\_name* and *macro=name* operands  
 24204 on the command line, expecting that all of the macros are processed before any of the targets are  
 24205 dealt with. Conforming applications do not do this, although some backwards-compatibility  
 24206 support may be included in some implementations.

24207 The following characters in filenames may give trouble: '=', ':', '\', '\'', and '@'. For  
 24208 inference rules, the description of \$< and \$? seem similar. However, an example shows the

24209 minor difference. In a makefile containing:

24210 `foo.o: foo.h`

24211 if **foo.h** is newer than **foo.o**, yet **foo.c** is older than **foo.o**, the built-in rule to make **foo.o** from  
 24212 **foo.c** is used, with \$< equal to **foo.c** and \$? equal to **foo.h**. If **foo.c** is also newer than **foo.o**, \$< is  
 24213 equal to **foo.c** and \$? is equal to **foo.h foo.c**.

#### 24214 EXAMPLES

24215 1. The following command:

24216 `make`

24217 makes the first target found in the makefile.

24218 2. The following command:

24219 `make junk`

24220 makes the target **junk**.

24221 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in  
 24222 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:

24223 `pgm: a.o b.o`

24224 `c99 a.o b.o -o pgm`

24225 `a.o: incl.h a.c`

24226 `c99 -c a.c`

24227 `b.o: incl.h b.c`

24228 `c99 -c b.c`

24229 4. An example for making optimized **.o** files from **.c** files is:

24230 `.c.o:`

24231 `c99 -c -O $*.c`

24232 or:

24233 `.c.o:`

24234 `c99 -c -O $<`

24235 5. The most common use of the archive interface follows. Here, it is assumed that the source  
 24236 files are all C-language source:

24237 `lib: lib(file1.o) lib(file2.o) lib(file3.o)`

24238 `@echo lib is now up-to-date`

24239 The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib**.

24240 The treatment of escaped <newline>s throughout the makefile is historical practice. For  
 24241 example, the inference rule:

24242 `.c.o\`

24243 `:`

24244 works, and the macro:

24245 `f= bar baz\`

24246 `biz`

24247 `a:`

24248 `echo ==$f==`

```

24249 echoes "==bar baz biz==".
24250 If $? were:
24251 /usr/include/stdio.h /usr/include/unistd.h foo.h
24252 then $(?D) would be:
24253 /usr/include /usr/include .
24254 and $(?F) would be:
24255 stdio.h unistd.h foo.h
24256 6. The contents of the built-in rules can be viewed by running:
24257 make -p -f /dev/null 2>/dev/null

```

#### 24258 RATIONALE

24259 The *make* utility described in this volume of IEEE Std 1003.1-2001 is intended to provide the  
 24260 means for changing portable source code into executables that can be run on an  
 24261 IEEE Std 1003.1-2001-conforming system. It reflects the most common features present in  
 24262 System V and BSD *makes*.

24263 Historically, the *make* utility has been an especially fertile ground for vendor and research  
 24264 organization-specific syntax modifications and extensions. Examples include:

- 24265 • Syntax supporting parallel execution (such as from various multi-processor vendors, GNU,  
 24266 and others)
- 24267 • Additional “operators” separating targets and their prerequisites (System V, BSD, and  
 24268 others)
- 24269 • Specifying that command lines containing the strings “\${MAKE}” and “\$(MAKE)” are  
 24270 executed when the `-n` option is specified (GNU and System V)
- 24271 • Modifications of the meaning of internal macros when referencing libraries (BSD and others)
- 24272 • Using a single instance of the shell for all of the command lines of the target (BSD and others)
- 24273 • Allowing spaces as well as tabs to delimit command lines (BSD)
- 24274 • Adding C preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and  
 24275 others)
- 24276 • Remote execution of command lines (Sprite and others)
- 24277 • Specifying additional special targets (BSD, System V, and most others)

24278 Additionally, many vendors and research organizations have rethought the basic concepts of  
 24279 *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of  
 24280 *make* fulfills the needs of a different community of users; it is unreasonable for this volume of  
 24281 IEEE Std 1003.1-2001 to require behavior that would be incompatible (and probably inferior) to  
 24282 historical practice for such a community.

24283 In similar circumstances, when the industry has enough sufficiently incompatible formats as to  
 24284 make them irreconcilable, this volume of IEEE Std 1003.1-2001 has followed one or both of two  
 24285 courses of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line  
 24286 options have been provided to select the desired behavior (*grep*, *od*, and *pax*).

24287 Because the syntax specified for the *make* utility is, by and large, a subset of the syntaxes  
 24288 accepted by almost all versions of *make*, it was decided that it would be counter-productive to  
 24289 change the name. And since the makefile itself is a basic unit of portability, it would not be

completely effective to reserve a new option letter, such as *make -P*, to achieve the portable behavior. Therefore, the special target **.POSIX** was added to the makefile, allowing users to specify “standard” behavior. This special target does not preclude extensions in the *make* utility, nor does it preclude such extensions being used by the makefile specifying the target; it does, however, preclude any extensions from being applied that could alter the behavior of previously valid syntax; such extensions must be controlled via command line options or new special targets. It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to guarantee that they are not affected by local extensions.

The portable version of *make* described in this reference page is not intended to be the state-of-the-art software generation tool and, as such, some newer and more leading-edge features have not been included. An attempt has been made to describe the portable makefile in a manner that does not preclude such extensions as long as they do not disturb the portable behavior described here.

When the **-n** option is specified, it is always added to **MAKEFLAGS**. This allows a recursive *make -n target* to be used to see all of the action that would be taken to update *target*.

The definition of **MAKEFLAGS** allows both the System V letter string and the BSD command line formats. The two formats are sufficiently different to allow implementations to support both without ambiguity.

Early proposals stated that an “unquoted” number sign was treated as the start of a comment. The *make* utility does not pay any attention to quotes. A number sign starts a comment regardless of its surroundings.

The text about “other implementation-defined pathnames may also be tried” in addition to **./makefile** and **./Makefile** is to allow such extensions as **SCCS/s.Makefile** and other variations. It was made an implementation-defined requirement (as opposed to unspecified behavior) to highlight surprising implementations that might select something unexpected like **/etc/Makefile**. XSI-conformant systems also try **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and **SCCS/s.Makefile**.

Early proposals contained the macro **NPROC** as a means of specifying that *make* should use *n* processes to do the work required. While this feature is a valuable extension for many systems, it is not common usage and could require other non-trivial extensions to makefile syntax. This extension is not required by this volume of IEEE Std 1003.1-2001, but could be provided as a compatible extension. The macro **PARALLEL** is used by some historical systems with essentially the same meaning (but without using a name that is a common system limit value). It is suggested that implementors recognize the existing use of **NPROC** and/or **PARALLEL** as extensions to *make*.

The default rules are based on System V. The default **CC=** value is *c99* instead of *cc* because this volume of IEEE Std 1003.1-2001 does not standardize the utility named *cc*. Thus, every conforming application would be required to define **CC=c99** to expect to run. There is no advantage conferred by the hope that the makefile might hit the “preferred” compiler because this cannot be guaranteed to work. Also, since the portable makescript can only use the *c99* options, no advantage is conferred in terms of what the script can do. It is a quality-of-implementation issue as to whether *c99* is as valuable as *cc*.

The **-d** option to *make* is frequently used to produce debugging information, but is too implementation-defined to add to this volume of IEEE Std 1003.1-2001.

The **-p** option is not passed in **MAKEFLAGS** on most historical implementations and to change this would cause many implementations to break without sufficiently increased portability.

Commands that begin with a plus sign ('+') are executed even if the **-n** option is present. Based on the GNU version of *make*, the behavior of **-n** when the plus-sign prefix is encountered has been extended to apply to **-q** and **-t** as well. However, the System V convention of forcing command execution with **-n** when the command line of a target contains either of the strings "\$ (MAKE)" or "\${MAKE}" has not been adopted. This functionality appeared in early proposals, but the danger of this approach was pointed out with the following example of a portion of a makefile:

```
subdir:
 cd subdir; rm all_the_files; $(MAKE)
```

The loss of the System V behavior in this case is well-balanced by the safety afforded to other makefiles that were not aware of this situation. In any event, the command line plus-sign prefix can provide the desired functionality.

The double colon in the target rule format is supported in BSD systems to allow more than one target line containing the same target name to have commands associated with it. Since this is not functionality described in the SVID or XPG3 it has been allowed as an extension, but not mandated.

The default rules are provided with text specifying that the built-in rules shall be the same as if the listed set were used. The intent is that implementations should be able to use the rules without change, but will be allowed to alter them in ways that do not affect the primary behavior.

The best way to provide portable makefiles is to include all of the rules needed in the makefile itself. The rules provided use only features provided by other portions of this volume of IEEE Std 1003.1-2001. The default rules include rules for optional commands in this volume of IEEE Std 1003.1-2001. Only rules pertaining to commands that are provided are needed in the default set of an implementation.

One point of discussion was whether to drop the default rules list from this volume of IEEE Std 1003.1-2001. They provide convenience, but do not enhance portability of applications. The prime benefit is in portability of users who wish to type *make command* and have the command build from a **command.c** file.

The historical *MAKESHELL* feature was omitted. In some implementations it is used to let a user override the shell to be used to run *make* commands. This was confusing; for a portable *make*, the shell should be chosen by the makefile writer or specified on the *make* command line and not by a user running *make*.

The *make* utilities in most historical implementations process the prerequisites of a target in left-to-right order, and the makefile format requires this. It supports the standard idiom used in many makefiles that produce yacc programs; for example:

```
foo: y.tab.o lex.o main.o
 $(CC) $(CFLAGS) -o $@ t.tab.o lex.o main.o
```

In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct **y.tab.h**. Although there may be better ways to express this relationship, it is widely used historically. Implementations that desire to update prerequisites in parallel should require an explicit extension to *make* or the makefile format to accomplish it, as described previously.

The algorithm for determining a new entry for target rules is partially unspecified. Some historical *makes* allow blank, empty, or comment lines within the collection of commands marked by leading <tab>s. A conforming makefile must ensure that each command starts with a <tab>, but implementations are free to ignore blank, empty, and comment lines without triggering the start of a new entry.

The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it is required to resend itself the signal it received so that it exits with a status that reflects the signal. The results from SIGQUIT are partially unspecified because, on systems that create **core** files upon receipt of SIGQUIT, the **core** from *make* would conflict with a **core** file from the command that was running when the SIGQUIT arrived. The main concern was to prevent damaged files from appearing up-to-date when *make* is rerun.

The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites; it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of targets than for the entire makefile. These extensions to *make* in System V were made to match historical practice from the BSD *make*.

Macros are not exported to the environment of commands to be run. This was never the case in any historical *make* and would have serious consequences. The environment is the same as the environment to *make* except that **MAKEFLAGS** and macros defined on the *make* command line are added.

Some implementations do not use *system()* for all command lines, as required by the portable makefile format; as a performance enhancement, they select lines without shell metacharacters for direct execution by *execve()*. There is no requirement that *system()* be used specifically, but merely that the same results be achieved. The metacharacters typically used to bypass the direct *execve()* execution have been any of:

= | ^ ( ) ; & < > \* ? [ ] : \$ \ ' " \ \n

The default in some advanced versions of *make* is to group all the command lines for a target and execute them using a single shell invocation; the System V method is to pass each line individually to a separate shell. The single-shell method has the advantages in performance and the lack of a requirement for many continued lines. However, converting to this newer method has caused portability problems with many historical makefiles, so the behavior with the POSIX makefile is specified to be the same as that of System V. It is suggested that the special target **.ONESHELL** be used as an implementation extension to achieve the single-shell grouping for a target or group of targets.

Novice users of *make* have had difficulty with the historical need to start commands with a <tab>. Since it is often difficult to discern differences between <tab>s and <space>s on terminals or printed listings, confusing bugs can arise. In early proposals, an attempt was made to correct this problem by allowing leading <blank>s instead of <tab>s. However, implementors reported many makefiles that failed in subtle ways following this change, and it is difficult to implement a *make* that unambiguously can differentiate between macro and command lines. There is extensive historical practice of allowing leading spaces before macro definitions. Forcing macro lines into column 1 would be a significant backwards-compatibility problem for some makefiles. Therefore, historical practice was restored.

The System V INCLUDE feature was considered, but not included. This would treat a line that began in the first column and contained INCLUDE <filename> as an indication to read <filename> at that point in the makefile. This is difficult to use in a portable way, and it raises concerns about nesting levels and diagnostics. System V, BSD, GNU, and others have used different methods for including files.

The System V dynamic dependency feature was not included. It would support:



24430 cat: \$\$@.c

24431 that would expand to;

24432 cat: cat.c

24433 This feature exists only in the new version of System V *make* and, while useful, is not in wide  
24434 usage. This means that macros are expanded twice for prerequisites: once at makefile parse time  
24435 and once at target update time.

24436 Consideration was given to adding metarules to the POSIX *make*. This would make **%o: %c** the  
24437 same as **.c.o:**. This is quite useful and available from some vendors, but it would cause too many  
24438 changes to this *make* to support. It would have introduced rule chaining and new substitution  
24439 rules. However, the rules for target names have been set to reserve the '%' and '"' characters.  
24440 These are traditionally used to implement metarules and quoting of target names, respectively.  
24441 Implementors are strongly encouraged to use these characters only for these purposes.

24442 A request was made to extend the suffix delimiter character from a period to any character. The  
24443 metarules feature in newer *makes* solves this problem in a more general way. This volume of  
24444 IEEE Std 1003.1-2001 is staying with the more conservative historical definition.

24445 The standard output format for the **-p** option is not described because it is primarily a  
24446 debugging option and because the format is not generally useful to programs. In historical  
24447 implementations the output is not suitable for use in generating makefiles. The **-p** format has  
24448 been variable across historical implementations. Therefore, the definition of **-p** was only to  
24449 provide a consistently named option for obtaining *make* script debugging information.

24450 Some historical implementations have not cleared the suffix list with **-r**.

24451 Implementations should be aware that some historical applications have intermixed *target\_name*  
24452 and *macro=value* operands on the command line, expecting that all of the macros are processed  
24453 before any of the targets are dealt with. Conforming applications do not do this, but some  
24454 backwards-compatibility support may be warranted.

24455 Empty inference rules are specified with a semicolon command rather than omitting all  
24456 commands, as described in an early proposal. The latter case has no traditional meaning and is  
24457 reserved for implementation extensions, such as in GNU *make*.

## 24458 FUTURE DIRECTIONS

24459 None.

## 24460 SEE ALSO

24461 Chapter 2 (on page 29), *ar*, *c99*, *get*, *lex*, *sccs*, *sh*, *yacc*, the System Interfaces volume of  
24462 IEEE Std 1003.1-2001, *exec*, *system()*

## 24463 CHANGE HISTORY

24464 First released in Issue 2.

### 24465 Issue 5

24466 The FUTURE DIRECTIONS section is added.

### 24467 Issue 6

24468 This utility is marked as part of the Software Development Utilities option.

24469 The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the  
24470 SPECIAL TARGETS section.

24471 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from  
24472 “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of  
24473 *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.

- 24474 It is specified whether the command line is related to the makefile or to the *make* command, and  
24475 the macro processing rules are updated to align with the IEEE P1003.2b draft standard.
- 24476 The normative text is reworded to avoid use of the term “must” for application requirements.
- 24477 PASC Interpretation 1003.2 #193 is applied.

24478 **NAME**

24479           man — display system documentation

24480 **SYNOPSIS**24481           man [-k] *name*...24482 **DESCRIPTION**

24483           The *man* utility shall write information about each of the *name* operands. If *name* is the name of a  
 24484           standard utility, *man* at a minimum shall write a message describing the syntax used by the  
 24485           standard utility, its options, and operands. If more information is available, the *man* utility shall  
 24486           provide it in an implementation-defined manner.

24487           An implementation may provide information for values of *name* other than the standard utilities.  
 24488           Standard utilities that are listed as optional and that are not supported by the implementation  
 24489           either shall cause a brief message indicating that fact to be displayed or shall cause a full display  
 24490           of information as described previously.

24491 **OPTIONS**

24492           The *man* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 24493           12.2, Utility Syntax Guidelines.

24494           The following option shall be supported:

24495           **-k**       Interpret *name* operands as keywords to be used in searching a utilities summary  
 24496                       database that contains a brief purpose entry for each standard utility and write lines  
 24497                       from the summary database that match any of the keywords. The keyword search shall  
 24498                       produce results that are the equivalent of the output of the following command:

```
24499 grep -Ei '
24500 name
24501 name
24502 ...
24503 ' summary-database
```

24504           This assumes that the *summary-database* is a text file with a single entry per line; this  
 24505           organization is not required and the example using *grep -Ei* is merely illustrative of the  
 24506           type of search intended. The purpose entry to be included in the database shall consist  
 24507           of a terse description of the purpose of the utility.

24508 **OPERANDS**

24509           The following operand shall be supported:

24510           *name*       A keyword or the name of a standard utility. When **-k** is not specified and *name*  
 24511                       does not represent one of the standard utilities, the results are unspecified.

24512 **STDIN**

24513           Not used.

24514 **INPUT FILES**

24515           None.

24516 **ENVIRONMENT VARIABLES**24517           The following environment variables shall affect the execution of *man*:

24518           **LANG**       Provide a default value for the internationalization variables that are unset or null.  
 24519                       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 24520                       Internationalization Variables for the precedence of internationalization variables  
 24521                       used to determine the values of locale categories.)

24522 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 24523 internationalization variables.

24524 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 24525 characters (for example, single-byte as opposed to multi-byte characters in  
 24526 arguments and in the summary database). The value of *LC\_CTYPE* need not affect  
 24527 the format of the information written about the *name* operands.

24528 *LC\_MESSAGES*  
 24529 Determine the locale that should be used to affect the format and contents of  
 24530 diagnostic messages written to standard error and informative messages written to  
 24531 standard output.

24532 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

24533 *PAGER* Determine an output filtering command for writing the output to a terminal. Any  
 24534 string acceptable as a *command\_string* operand to the *sh -c* command shall be valid.  
 24535 When standard output is a terminal device, the reference page output shall be  
 24536 piped through the command. If the *PAGER* variable is null or not set, the  
 24537 command shall be either *more* or another paginator utility documented in the  
 24538 system documentation.

24539 **ASYNCHRONOUS EVENTS**

24540 Default.

24541 **STDOUT**  
 24542 The *man* utility shall write text describing the syntax of the utility *name*, its options and its  
 24543 operands, or, when *-k* is specified, lines from the summary database. The format of this text is  
 24544 implementation-defined.

24545 **STDERR**  
 24546 The standard error shall be used only for diagnostic messages.

24547 **OUTPUT FILES**  
 24548 None.

24549 **EXTENDED DESCRIPTION**  
 24550 None.

24551 **EXIT STATUS**  
 24552 The following exit values shall be returned:  
 24553 0 Successful completion.  
 24554 >0 An error occurred.

24555 **CONSEQUENCES OF ERRORS**  
 24556 Default.

24557 **APPLICATION USAGE**  
 24558 None.

24559 **EXAMPLES**  
 24560 None.

24561 **RATIONALE**  
 24562 It is recognized that the *man* utility is only of minimal usefulness as specified. The opinion of the  
 24563 standard developers was strongly divided as to how much or how little information *man* should  
 24564 be required to provide. They considered, however, that the provision of some portable way of  
 24565 accessing documentation would aid user portability. The arguments against a fuller

24566 specification were:

- 24567 • Large quantities of documentation should not be required on a system that does not have
- 24568 excess disk space.
- 24569 • The current manual system does not present information in a manner that greatly aids user
- 24570 portability.
- 24571 • A “better help system” is currently an area in which vendors feel that they can add value to
- 24572 their POSIX implementations.

24573 The **-f** option was considered, but due to implementation differences, it was not included in this

24574 volume of IEEE Std 1003.1-2001.

24575 The description was changed to be more specific about what has to be displayed for a utility.

24576 The standard developers considered it insufficient to allow a display of only the synopsis

24577 without giving a short description of what each option and operand does.

24578 The “purpose” entry to be included in the database can be similar to the section title (less the

24579 numeric prefix) from this volume of IEEE Std 1003.1-2001 for each utility. These titles are similar

24580 to those used in historical systems for this purpose.

24581 See *mailx* for rationale concerning the default paginator.

24582 The caveat in the *LC\_CTYPE* description was added because it is not a requirement that an

24583 implementation provide reference pages for all of its supported locales on each system;

24584 changing *LC\_CTYPE* does not necessarily translate the reference page into another language.

24585 This is equivalent to the current state of *LC\_MESSAGES* in IEEE Std 1003.1-2001—locale-specific

24586 messages are not yet a requirement.

24587 The historical *MANPATH* variable is not included in POSIX because no attempt is made to

24588 specify naming conventions for reference page files, nor even to mandate that they are files at

24589 all. On some implementations they could be a true database, a hypertext file, or even fixed

24590 strings within the *man* executable. The standard developers considered the portability of

24591 reference pages to be outside their scope of work. However, users should be aware that

24592 *MANPATH* is implemented on a number of historical systems and that it can be used to tailor

24593 the search pattern for reference pages from the various categories (utilities, functions, file

24594 formats, and so on) when the system administrator reveals the location and conventions for

24595 reference pages on the system.

24596 The keyword search can rely on at least the text of the section titles from these utility

24597 descriptions, and the implementation may add more keywords. The term “section titles” refers

24598 to the strings such as:

24599 *man* — Display system documentation

24600 *ps* — Report process status

24601 **FUTURE DIRECTIONS**

24602 None.

24603 **SEE ALSO**

24604 *more*

24605 **CHANGE HISTORY**

24606 First released in Issue 4.

24607 **Issue 5**

24608 The FUTURE DIRECTIONS section is added.

24609 **NAME**

24610           mesg — permit or deny messages

24611 **SYNOPSIS**

24612 UP       mesg [y|n]

24613

24614 **DESCRIPTION**

24615       The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or  
 24616       other utilities to a terminal device. The terminal device affected shall be determined by searching  
 24617       for the first terminal in the sequence of devices associated with standard input, standard output,  
 24618       and standard error, respectively. With no arguments, *mesg* shall report the current state without  
 24619       changing it. Processes with appropriate privileges may be able to send messages to the terminal  
 24620       independent of the current state.

24621 **OPTIONS**

24622       None.

24623 **OPERANDS**

24624       The following operands shall be supported in the POSIX locale:

24625       y           Grant permission to other users to send messages to the terminal device.

24626       n           Deny permission to other users to send messages to the terminal device.

24627 **STDIN**

24628       Not used.

24629 **INPUT FILES**

24630       None.

24631 **ENVIRONMENT VARIABLES**24632       The following environment variables shall affect the execution of *mesg*:

24633       LANG       Provide a default value for the internationalization variables that are unset or null.  
 24634       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 24635       Internationalization Variables for the precedence of internationalization variables  
 24636       used to determine the values of locale categories.)

24637       LC\_ALL     If set to a non-empty string value, override the values of all the other  
 24638       internationalization variables.

24639       LC\_CTYPE   Determine the locale for the interpretation of sequences of bytes of text data as  
 24640       characters (for example, single-byte as opposed to multi-byte characters in  
 24641       arguments).

24642       LC\_MESSAGES

24643       Determine the locale that should be used to affect the format and contents of  
 24644       diagnostic messages written (by *mesg*) to standard error.

24645 XSI       NLSPATH   Determine the location of message catalogs for the processing of LC\_MESSAGES.

24646 **ASYNCHRONOUS EVENTS**

24647       Default.

24648 **STDOUT**24649       If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.

24650 **STDERR**

24651           The standard error shall be used only for diagnostic messages.

24652 **OUTPUT FILES**

24653           None.

24654 **EXTENDED DESCRIPTION**

24655           None.

24656 **EXIT STATUS**

24657           The following exit values shall be returned:

24658           0   Receiving messages is allowed.

24659           1   Receiving messages is not allowed.

24660           >1  An error occurred.

24661 **CONSEQUENCES OF ERRORS**

24662           Default.

24663 **APPLICATION USAGE**

24664           The mechanism by which the message status of the terminal is changed is unspecified.  
24665           Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has  
24666           successfully completed. These actions may include, but are not limited to: another invocation of  
24667           the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or  
24668           *chmod()* function, and so on.

24669 **EXAMPLES**

24670           None.

24671 **RATIONALE**

24672           The terminal changed by *mesg* is that associated with the standard input, output, or error, rather  
24673           than the controlling terminal for the session. This is because users logged in more than once  
24674           should be able to change any of their login terminals without having to stop the job running in  
24675           those sessions. This is not a security problem involving the terminals of other users because  
24676           appropriate privileges would be required to affect the terminal of another user.

24677           The method of checking each of the first three file descriptors in sequence until a terminal is  
24678           found was adopted from System V.

24679           The file */dev/tty* is not specified for the terminal device because it was thought to be too  
24680           restrictive. Typical environment changes for the *n* operand are that write permissions are  
24681           removed for *others* and *group* from the appropriate device. It was decided to leave the actual  
24682           description of what is done as unspecified because of potential differences between  
24683           implementations.

24684           The format for standard output is unspecified because of differences between historical  
24685           implementations. This output is generally not useful to shell scripts (they can use the exit  
24686           status), so exact parsing of the output is unnecessary.

24687 **FUTURE DIRECTIONS**

24688           None.

24689 **SEE ALSO**

24690           *talk*, *write*



24691 **CHANGE HISTORY**

24692 First released in Issue 2.

24693 **Issue 6**

24694 This utility is marked as part of the User Portability Utilities option.

24695 **NAME**

24696       mkdir — make directories

24697 **SYNOPSIS**

24698       mkdir [-p] [-m *mode*] *dir*...

24699 **DESCRIPTION**

24700       The *mkdir* utility shall create the directories specified by the operands, in the order specified.

24701       For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir()* function  
24702       defined in the System Interfaces volume of IEEE Std 1003.1-2001, called with the following  
24703       arguments:

- 24704       1. The *dir* operand is used as the *path* argument.
- 24705       2. The value of the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO is used as  
24706       the *mode* argument. (If the **-m** option is specified, the *mode* option-argument overrides this  
24707       default.)

24708 **OPTIONS**

24709       The *mkdir* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
24710       12.2, Utility Syntax Guidelines.

24711       The following options shall be supported:

24712       **-m** *mode*       Set the file permission bits of the newly-created directory to the specified *mode*  
24713       value. The *mode* option-argument shall be the same as the *mode* operand defined  
24714       for the *chmod* utility. In the *symbolic\_mode* strings, the *op* characters '+' and '-'  
24715       shall be interpreted relative to an assumed initial mode of *a=rwx*; '+' shall add  
24716       permissions to the default mode, '-' shall delete permissions from the default  
24717       mode.

24718       **-p**               Create any missing intermediate pathname components.  
24719       For each *dir* operand that does not name an existing directory, effects equivalent to  
24720       those caused by the following command shall occur:

24721       mkdir -p -m \$(umask -S),u+wx \$(dirname *dir*) &&  
24722       mkdir [-m *mode*] *dir*

24723       where the **-m** *mode* option represents that option supplied to the original  
24724       invocation of *mkdir*, if any.

24725       Each *dir* operand that names an existing directory shall be ignored without error.

24726 **OPERANDS**

24727       The following operand shall be supported:

24728       *dir*             A pathname of a directory to be created.

24729 **STDIN**

24730       Not used.

24731 **INPUT FILES**

24732       None.

24733 **ENVIRONMENT VARIABLES**

24734       The following environment variables shall affect the execution of *mkdir*:

24735       *LANG*            Provide a default value for the internationalization variables that are unset or null.  
24736       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
24737       Internationalization Variables for the precedence of internationalization variables

|       |                               |                                                                                                       |
|-------|-------------------------------|-------------------------------------------------------------------------------------------------------|
| 24738 |                               | used to determine the values of locale categories.)                                                   |
| 24739 | <i>LC_ALL</i>                 | If set to a non-empty string value, override the values of all the other                              |
| 24740 |                               | internationalization variables.                                                                       |
| 24741 | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as                     |
| 24742 |                               | characters (for example, single-byte as opposed to multi-byte characters in                           |
| 24743 |                               | arguments).                                                                                           |
| 24744 | <i>LC_MESSAGES</i>            |                                                                                                       |
| 24745 |                               | Determine the locale that should be used to affect the format and contents of                         |
| 24746 |                               | diagnostic messages written to standard error.                                                        |
| 24747 | XSI <i>NLSPATH</i>            | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                 |
| 24748 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                       |
| 24749 |                               | Default.                                                                                              |
| 24750 | <b>STDOUT</b>                 |                                                                                                       |
| 24751 |                               | Not used.                                                                                             |
| 24752 | <b>STDERR</b>                 |                                                                                                       |
| 24753 |                               | The standard error shall be used only for diagnostic messages.                                        |
| 24754 | <b>OUTPUT FILES</b>           |                                                                                                       |
| 24755 |                               | None.                                                                                                 |
| 24756 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                       |
| 24757 |                               | None.                                                                                                 |
| 24758 | <b>EXIT STATUS</b>            |                                                                                                       |
| 24759 |                               | The following exit values shall be returned:                                                          |
| 24760 | 0                             | All the specified directories were created successfully or the <b>-p</b> option was specified and all |
| 24761 |                               | the specified directories now exist.                                                                  |
| 24762 | >0                            | An error occurred.                                                                                    |
| 24763 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                       |
| 24764 |                               | Default.                                                                                              |
| 24765 | <b>APPLICATION USAGE</b>      |                                                                                                       |
| 24766 |                               | The default file mode for directories is <i>a=rwx</i> (777 on most systems) with selected permissions |
| 24767 |                               | removed in accordance with the file mode creation mask. For intermediate pathname                     |
| 24768 |                               | components created by <i>mkdir</i> , the mode is the default modified by <i>u+wx</i> so that the      |
| 24769 |                               | subdirectories can always be created regardless of the file mode creation mask; if different          |
| 24770 |                               | ultimate permissions are desired for the intermediate directories, they can be changed                |
| 24771 |                               | afterwards with <i>chmod</i> .                                                                        |
| 24772 |                               | Note that some of the requested directories may have been created even if an error occurs.            |
| 24773 | <b>EXAMPLES</b>               |                                                                                                       |
| 24774 |                               | None.                                                                                                 |
| 24775 | <b>RATIONALE</b>              |                                                                                                       |
| 24776 |                               | The System V <b>-m</b> option was included to control the file mode.                                  |
| 24777 |                               | The System V <b>-p</b> option was included to create any needed intermediate directories and to       |
| 24778 |                               | complement the functionality provided by <i>rmdir</i> for removing directories in the path prefix as  |
| 24779 |                               | they become empty. Because no error is produced if any path component already exists, the <b>-p</b>   |
| 24780 |                               | option is also useful to ensure that a particular directory exists.                                   |

24781           The functionality of *mkdir* is described substantially through a reference to the *mkdir()* function  
24782           in the System Interfaces volume of IEEE Std 1003.1-2001. For example, by default, the mode of  
24783           the directory is affected by the file mode creation mask in accordance with the specified  
24784           behavior of the *mkdir()* function. In this way, there is less duplication of effort required for  
24785           describing details of the directory creation.

24786 **FUTURE DIRECTIONS**

24787           None.

24788 **SEE ALSO**

24789           *chmod*, *rm*, *rmdir*, *umask*, the System Interfaces volume of IEEE Std 1003.1-2001, *mkdir()*

24790 **CHANGE HISTORY**

24791           First released in Issue 2.

24792 **Issue 5**

24793           The FUTURE DIRECTIONS section is added.

24794 **NAME**

24795 mkfifo — make FIFO special files

24796 **SYNOPSIS**24797 mkfifo [-m *mode*] *file*...24798 **DESCRIPTION**24799 The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order  
24800 specified.24801 For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo()* function  
24802 defined in the System Interfaces volume of IEEE Std 1003.1-2001, called with the following  
24803 arguments:

- 24804 1. The *file* operand is used as the *path* argument.
- 24805 2. The value of the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP,  
24806 S\_IROTH, and S\_IWOTH is used as the *mode* argument. (If the **-m** option is specified, the  
24807 value of the *mkfifo()* *mode* argument is unspecified, but the FIFO shall at no time have  
24808 permissions less restrictive than the **-m mode** option-argument.)

24809 **OPTIONS**24810 The *mkfifo* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
24811 12.2, Utility Syntax Guidelines.

24812 The following option shall be supported:

24813 **-m mode** Set the file permission bits of the newly-created FIFO to the specified *mode* value.  
 24814 The *mode* option-argument shall be the same as the *mode* operand defined for the  
 24815 *chmod* utility. In the *symbolic\_mode* strings, the *op* characters '+' and '-' shall be  
 24816 interpreted relative to an assumed initial mode of *a=rw*.

24817 **OPERANDS**

24818 The following operand shall be supported:

24819 *file* A pathname of the FIFO special file to be created.24820 **STDIN**

24821 Not used.

24822 **INPUT FILES**

24823 None.

24824 **ENVIRONMENT VARIABLES**24825 The following environment variables shall affect the execution of *mkfifo*:

24826 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 24827 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 24828 Internationalization Variables for the precedence of internationalization variables  
 24829 used to determine the values of locale categories.)

24830 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 24831 internationalization variables.

24832 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 24833 characters (for example, single-byte as opposed to multi-byte characters in  
 24834 arguments).

24835 **LC\_MESSAGES**

24836 Determine the locale that should be used to affect the format and contents of  
 24837 diagnostic messages written to standard error.

24838 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 24839 ASYNCHRONOUS EVENTS

24840 Default.

## 24841 STDOUT

24842 Not used.

## 24843 STDERR

24844 The standard error shall be used only for diagnostic messages.

## 24845 OUTPUT FILES

24846 None.

## 24847 EXTENDED DESCRIPTION

24848 None.

## 24849 EXIT STATUS

24850 The following exit values shall be returned:

24851 0 All the specified FIFO special files were created successfully.

24852 >0 An error occurred.

## 24853 CONSEQUENCES OF ERRORS

24854 Default.

## 24855 APPLICATION USAGE

24856 None.

## 24857 EXAMPLES

24858 None.

## 24859 RATIONALE

24860 This utility was added to permit shell applications to create FIFO special files.

24861 The **-m** option was added to control the file mode, for consistency with the similar functionality  
24862 provided by the *mkdir* utility.

24863 Early proposals included a **-p** option similar to the *mkdir -p* option that created intermediate  
24864 directories leading up to the FIFO specified by the final component. This was removed because  
24865 it is not commonly needed and is not common practice with similar utilities.

24866 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo()* function  
24867 in the System Interfaces volume of IEEE Std 1003.1-2001. For example, by default, the mode of  
24868 the FIFO file is affected by the file mode creation mask in accordance with the specified behavior  
24869 of the *mkfifo()* function. In this way, there is less duplication of effort required for describing  
24870 details of the file creation.

## 24871 FUTURE DIRECTIONS

24872 None.

## 24873 SEE ALSO

24874 *chmod*, *umask*, the System Interfaces volume of IEEE Std 1003.1-2001, *mkfifo()*

## 24875 CHANGE HISTORY

24876 First released in Issue 3.

24877 **NAME**24878        *more* — display files on a page-by-page basis24879 **SYNOPSIS**24880 UP        *more* [-ceisu] [-n *number*] [-p *command*] [-t *tagstring*] [*file* ...]

24881

24882 **DESCRIPTION**

24883        The *more* utility shall read files and either write them to the terminal on a page-by-page basis or  
 24884        filter them to standard output. If standard output is not a terminal device, all input files shall be  
 24885        copied to standard output in their entirety, without modification, except as specified for the **-s**  
 24886        option. If standard output is a terminal device, the files shall be written a number of lines (one  
 24887        screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION  
 24888        section.

24889        Certain block-mode terminals do not have all the capabilities necessary to support the complete  
 24890        *more* definition; they are incapable of accepting commands that are not terminated with a  
 24891        <newline>. Implementations that support such terminals shall provide an operating mode to  
 24892        *more* in which all commands can be terminated with a <newline> on those terminals. This mode:

- 24893        • Shall be documented in the system documentation
- 24894        • Shall, at invocation, inform the user of the terminal deficiency that requires the <newline>  
 24895        usage and provide instructions on how this warning can be suppressed in future invocations
- 24896        • Shall not be required for implementations supporting only fully capable terminals
- 24897        • Shall not affect commands already requiring <newline>s
- 24898        • Shall not affect users on the capable terminals from using *more* as described in this volume of  
 24899        IEEE Std 1003.1-2001

24900 **OPTIONS**

24901        The *more* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 24902        12.2, Utility Syntax Guidelines.

24903        The following options shall be supported:

- 24904        **-c**        If a screen is to be written that has no lines in common with the current screen, or  
 24905        *more* is writing its first screen, *more* shall not scroll the screen, but instead shall  
 24906        redraw each line of the screen in turn, from the top of the screen to the bottom. In  
 24907        addition, if *more* is writing its first screen, the screen shall be cleared. This option  
 24908        may be silently ignored on devices with insufficient terminal capabilities.
- 24909        **-e**        By default, *more* shall exit immediately after writing the last line of the last file in  
 24910        the argument list. If the **-e** option is specified:
  - 24911            1. If there is only a single file in the argument list and that file was completely  
 24912            displayed on a single screen, *more* shall exit immediately after writing the last  
 24913            line of that file.
  - 24914            2. Otherwise, *more* shall exit only after reaching end-of-file on the last file in the  
 24915            argument list twice without an intervening operation. See the EXTENDED  
 24916            DESCRIPTION section.
- 24917        **-i**        Perform pattern matching in searches without regard to case; see the Base  
 24918        Definitions volume of IEEE Std 1003.1-2001, Section 9.2, Regular Expression  
 24919        General Requirements.

24920       **-n number**   Specify the number of lines per screenful. The *number* argument is a positive  
 24921                   decimal integer. The **-n** option shall override any values obtained from any other  
 24922                   source.

24923       **-p command** Each time a screen from a new file is displayed or redisplayed (including as a  
 24924                   result of *more* commands; for example, **:p**), execute the *more* command(s) in the  
 24925                   command arguments in the order specified, as if entered by the user after the first  
 24926                   screen has been displayed. No intermediate results shall be displayed (that is, if the  
 24927                   command is a movement to a screen different from the normal first screen, only  
 24928                   the screen resulting from the command shall be displayed.) If any of the  
 24929                   commands fail for any reason, an informational message to this effect shall be  
 24930                   written, and no further commands specified using the **-p** option shall be executed  
 24931                   for this file.

24932       **-s**               Behave as if consecutive empty lines were a single empty line.

24933       **-t tagstring** Write the screenful of the file containing the tag named by the *tagstring* argument.  
 24934                   See the *ctags* utility. The tags feature represented by **-t tagstring** and the **:t**  
 24935                   command is optional. It shall be provided on any system that also provides a  
 24936                   conforming implementation of *ctags*; otherwise, the use of **-t** produces undefined  
 24937                   results.

24938                   The filename resulting from the **-t** option shall be logically added as a prefix to the  
 24939                   list of command line files, as if specified by the user. If the tag named by the  
 24940                   *tagstring* argument is not found, it shall be an error, and *more* shall take no further  
 24941                   action.

24942                   If the tag specifies a line number, the first line of the display shall contain the  
 24943                   beginning of that line. If the tag specifies a pattern, the first line of the display shall  
 24944                   contain the beginning of the matching text from the first line of the file that  
 24945                   contains that pattern. If the line does not exist in the file or matching text is not  
 24946                   found, an informational message to this effect shall be displayed, and *more* shall  
 24947                   display the default screen as if **-t** had not been specified.

24948                   If both the **-t tagstring** and **-p command** options are given, the **-t tagstring** shall be  
 24949                   processed first; that is, the file and starting line for the display shall be as specified  
 24950                   by **-t**, and then the **-p more** command shall be executed. If the line (matching text)  
 24951                   specified by the **-t** command does not exist (is not found), no **-p more** command  
 24952                   shall be executed for this file at any time.

24953       **-u**               Treat a <backspace> as a printable control character, displayed as an  
 24954                   implementation-defined character sequence (see the EXTENDED DESCRIPTION  
 24955                   section), suppressing backspacing and the special handling that produces  
 24956                   underlined or standout mode text on some terminal types. Also, do not ignore a  
 24957                   <carriage-return> at the end of a line.

## 24958 OPERANDS

24959       The following operand shall be supported:

24960       *file*            A pathname of an input file. If no *file* operands are specified, the standard input  
 24961                   shall be used. If a *file* is '-', the standard input shall be read at that point in the  
 24962                   sequence.

## 24963 STDIN

24964       The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.



24965 **INPUT FILES**

24966 The input files being examined shall be text files. If standard output is a terminal, standard error  
 24967 shall be used to read commands from the user. If standard output is a terminal, standard error is  
 24968 not readable, and command input is needed, *more* may attempt to obtain user commands from  
 24969 the controlling terminal (for example, */dev/tty*); otherwise, *more* shall terminate with an error  
 24970 indicating that it was unable to read user commands. If standard output is not a terminal, no  
 24971 error shall result if standard error cannot be opened for reading.

24972 **ENVIRONMENT VARIABLES**

24973 The following environment variables shall affect the execution of *more*:

24974 **COLUMNS** Override the system-selected horizontal display line size. See the Base Definitions  
 24975 volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables for valid values  
 24976 and results when it is unset or null.

24977 **EDITOR** Used by the **v** command to select an editor. See the EXTENDED DESCRIPTION  
 24978 section.

24979 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 24980 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 24981 Internationalization Variables for the precedence of internationalization variables  
 24982 used to determine the values of locale categories.)

24983 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 24984 internationalization variables.

24985 **LC\_COLLATE**  
 24986 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 24987 character collating elements within regular expressions.

24988 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 24989 characters (for example, single-byte as opposed to multi-byte characters in  
 24990 arguments and input files) and the behavior of character classes within regular  
 24991 expressions.

24992 **LC\_MESSAGES**  
 24993 Determine the locale that should be used to affect the format and contents of  
 24994 diagnostic messages written to standard error and informative messages written to  
 24995 standard output.

24996 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

24997 **LINES** Override the system-selected vertical screen size, used as the number of lines in a  
 24998 screenful. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8,  
 24999 Environment Variables for valid values and results when it is unset or null. The **-n**  
 25000 option shall take precedence over the **LINES** variable for determining the number  
 25001 of lines in a screenful.

25002 **MORE** Determine a string containing options described in the OPTIONS section preceded  
 25003 with hyphens and <blank>-separated as on the command line. Any command line  
 25004 options shall be processed after those in the **MORE** variable, as if the command  
 25005 line were:

25006 *more \$MORE options operands*

25007 The **MORE** variable shall take precedence over the **TERM** and **LINES** variables for  
 25008 determining the number of lines in a screenful.

25009           **TERM**           Determine the name of the terminal type. If this variable is unset or null, an  
25010                               unspecified default terminal type is used.

## 25011 **ASYNCHRONOUS EVENTS**

25012           Default.

## 25013 **STDOUT**

25014           The standard output shall be used to write the contents of the input files.

## 25015 **STDERR**

25016           The standard error shall be used for diagnostic messages and user commands (see the INPUT  
25017           FILES section), and, if standard output is a terminal device, to write a prompting string. The  
25018           prompting string shall appear on the screen line below the last line of the file displayed in the  
25019           current screenful. The prompt shall contain the name of the file currently being examined and  
25020           shall contain an end-of-file indication and the name of the next file, if any, when prompting at  
25021           the end-of-file. If an error or informational message is displayed, it is unspecified whether it is  
25022           contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the  
25023           user shall be prompted for a continuation character, at which point another message or the user  
25024           prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether  
25025           informational messages are written for other user commands.

## 25026 **OUTPUT FILES**

25027           None.

## 25028 **EXTENDED DESCRIPTION**

25029           The following section describes the behavior of *more* when the standard output is a terminal  
25030           device. If the standard output is not a terminal device, no options other than **-s** shall have any  
25031           effect, and all input files shall be copied to standard output otherwise unmodified, at which time  
25032           *more* shall exit without further action.

25033           The number of lines available per screen shall be determined by the **-n** option, if present, or by  
25034           examining values in the environment (see the ENVIRONMENT VARIABLES section). If neither  
25035           method yields a number, an unspecified number of lines shall be used.

25036           The maximum number of lines written shall be one less than this number, because the screen  
25037           line after the last line written shall be used to write a user prompt and user input. If the number  
25038           of lines in the screen is less than two, the results are undefined. It is unspecified whether user  
25039           input is permitted to be longer than the remainder of the single line where the prompt has been  
25040           written.

25041           The number of columns available per line shall be determined by examining values in the  
25042           environment (see the ENVIRONMENT VARIABLES section), with a default value as described  
25043           in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

25044           Lines that are longer than the display shall be folded; the length at which folding occurs is  
25045           unspecified, but should be appropriate for the output device. Folding may occur between glyphs  
25046           of single characters that take up multiple display columns.

25047           When standard output is a terminal and **-u** is not specified, *more* shall treat <backspace>s and  
25048           <carriage-return>s specially:

- 25049           • A character, followed first by a sequence of *n* <backspace>s (where *n* is the same as the  
25050           number of column positions that the character occupies), then by *n* underscore characters  
25051           (' \_ '), shall cause that character to be written as underlined text, if the terminal type  
25052           supports that. The *n* underscore characters, followed first by *n* <backspace>s, then any  
25053           character with *n* column positions, shall also cause that character to be written as underlined  
25054           text, if the terminal type supports that.

- A sequence of *n* <backspace>s (where *n* is the same as the number of column positions that the previous character occupies) that appears between two identical printable characters shall cause the first of those two characters to be written as emboldened text (that is, visually brighter, standout mode, or inverse-video mode), if the terminal type supports that, and the second to be discarded. Immediately subsequent occurrences of <backspace> /character pairs for that same character shall also be discarded. (For example, the sequence "a\ba\ba\ba" is interpreted as a single emboldened 'a'.)
- The *more* utility shall logically discard all other <backspace>s from the line as well as the character which precedes them, if any.
- A <carriage-return> at the end of a line shall be ignored, rather than being written as a non-printable character, as described in the next paragraph.

It is implementation-defined how other non-printable characters are written. Implementations should use the same format that they use for the *ex print* command; see the OPTIONS section within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the number of columns on the display is less than the number of columns any single character in the line being displayed would occupy.

When each new file is displayed (or redisplayed), *more* shall write the first screen of the file. Once the initial screen has been written, *more* shall prompt for a user command. If the execution of the user command results in a screen that has lines in common with the current screen, and the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is unspecified whether the screen is scrolled or redrawn.

For all files but the last (including standard input if no file was specified, and for the last file as well, if the *-e* option was not specified), when *more* has written the last line in the file, *more* shall prompt for a user command. This prompt shall contain the name of the next file as well as an indication that *more* has reached end-of-file. If the user command is **f**, <control>-F, <space>, **j**, <newline>, **d**, <control>-D, or **s**, *more* shall display the next file. Otherwise, if displaying the last file, *more* shall exit. Otherwise, *more* shall execute the user command specified.

Several of the commands described in this section display a previous screen from the input stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is implementation-defined how much backwards motion is supported. If a command cannot be executed because of a limitation on backwards motion, an error message to this effect shall be displayed, the current screen shall not change, and the user shall be prompted for another command.

If a command cannot be performed because there are insufficient lines to display, *more* shall alert the terminal. If a command cannot be performed because there are insufficient lines to display or a / command fails: if the input is the standard input, the last screen in the file may be displayed; otherwise, the current file and screen shall not change, and the user shall be prompted for another command.

The interactive commands in the following sections shall be supported. Some commands can be preceded by a decimal integer, called *count* in the following descriptions. If not specified with the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular expression, as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions. The term “examine” is historical usage meaning “open the file for viewing”; for example, *more foo* would be expressed as examining file **foo**.

In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a line from the file being examined.

25102 In the following descriptions, the *current position* refers to two things:

- 25103 1. The position of the current line on the screen
- 25104 2. The line number (in the file) of the current line on the screen

25105 Usually, the line on the screen corresponding to the current position is the third line on the  
 25106 screen. If this is not possible (there are fewer than three lines to display or this is the first page of  
 25107 the file, or it is the last page of the file), then the current position is either the first or last line on  
 25108 the screen as described later.

## 25109 **Help**

25110 *Synopsis:*     h

25111 Write a summary of these commands and other implementation-defined commands. The  
 25112 behavior shall be as if the *more* utility were executed with the **-e** option on a file that contained  
 25113 the summary information. The user shall be prompted as described earlier in this section when  
 25114 end-of-file is reached. If the user command is one of those specified to continue to the next file,  
 25115 *more* shall return to the file and screen state from which the **h** command was executed.

## 25116 **Scroll Forward One Screenful**

25117 *Synopsis:*     [count] f  
 25118                 [count] <control>-F

25119 Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size,  
 25120 only the final screenful shall be written.

## 25121 **Scroll Backward One Screenful**

25122 *Synopsis:*     [count] b  
 25123                 [count] <control>-B

25124 Scroll backward *count* lines, with a default of one screenful (see the **-n** option). If *count* is more  
 25125 than the screen size, only the final screenful shall be written.

## 25126 **Scroll Forward One Line**

25127 *Synopsis:*     [count] <space>  
 25128                 [count] j  
 25129                 [count] <newline>

25130 Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for **j** and  
 25131 <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen  
 25132 size.

## 25133 **Scroll Backward One Line**

25134 *Synopsis:*     [count] k

25135 Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the  
 25136 screen size.

### 25137 **Scroll Forward One Half Screenful**

25138 *Synopsis:*     [*count*] d  
 25139                   [*count*] <control>-D

25140 Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it  
 25141 shall become the new default for subsequent **d**, <control>-D, and **u** commands.

### 25142 **Skip Forward One Line**

25143 *Synopsis:*     [*count*] s

25144 Display the screenful beginning with the line *count* lines after the last line on the current screen.  
 25145 If *count* would cause the current position to be such that less than one screenful would be  
 25146 written, the last screenful in the file shall be written.

### 25147 **Scroll Backward One Half Screenful**

25148 *Synopsis:*     [*count*] u  
 25149                   [*count*] <control>-U

25150 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it  
 25151 shall become the new default for subsequent **d**, <control>-D, **u**, and <control>-U commands.  
 25152 The entire *count* lines shall be written, even if *count* is more than the screen size.

### 25153 **Go to Beginning of File**

25154 *Synopsis:*     [*count*] g

25155 Display the screenful beginning with line *count*.

### 25156 **Go to End-of-File**

25157 *Synopsis:*     [*count*] G

25158 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the  
 25159 last screenful of the file.

### 25160 **Refresh the Screen**

25161 *Synopsis:*     r  
 25162                   <control>-L

25163 Refresh the screen.

### 25164 **Discard and Refresh**

25165 *Synopsis:*     R

25166 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered  
 25167 input shall not be discarded and the **R** command shall be equivalent to the **r** command.

25168 **Mark Position**25169 *Synopsis:*     *mletter*

25170 Mark the current position with the letter named by *letter*, where *letter* represents the name of one  
 25171 of the lowercase letters of the portable character set. When a new file is examined, all marks may  
 25172 be lost.

25173 **Return to Mark**25174 *Synopsis:*     '*letter*

25175 Return to the position that was previously marked with the letter named by *letter*, making that  
 25176 line the current position.

25177 **Return to Previous Position**25178 *Synopsis:*     ''

25179 Return to the position from which the last large movement command was executed (where a  
 25180 “large movement” is defined as any movement of more than a screenful of lines). If no such  
 25181 movements have been made, return to the beginning of the file.

25182 **Search Forward for Pattern**25183 *Synopsis:*     [*count*] / [!] *pattern* <newline>

25184 Display the screenful beginning with the *count*th line containing the pattern. The search shall  
 25185 start after the first line currently displayed. The null regular expression ('/' followed by a  
 25186 <newline>) shall repeat the search using the previous regular expression, with a default *count*. If  
 25187 the character '!' is included, the matching lines shall be those that do not contain the *pattern*. If  
 25188 no match is found for the *pattern*, a message to that effect shall be displayed.

25189 **Search Backward for Pattern**25190 *Synopsis:*     [*count*] ? [!] *pattern* <newline>

25191 Display the screenful beginning with the *count*th previous line containing the pattern. The  
 25192 search shall start on the last line before the first line currently displayed. The null regular  
 25193 expression ('?' followed by a <newline>) shall repeat the search using the previous regular  
 25194 expression, with a default *count*. If the character '!' is included, matching lines shall be those  
 25195 that do not contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall  
 25196 be displayed.

25197 **Repeat Search**25198 *Synopsis:*     [*count*] n

25199 Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last  
 25200 *pattern*, if the previous search was "/" or "?").

## 25201 **Repeat Search in Reverse**

25202 *Synopsis:*     [*count*]N

25203 Repeat the search in the opposite direction of the previous search for the *count*th line containing  
25204 the last *pattern* (or not containing the last *pattern*, if the previous search was `/! "` or `?! "`).

## 25205 **Examine New File**

25206 *Synopsis:*     :e [*filename*]<newline>

25207 Examine a new file. If the *filename* argument is not specified, the current file (see the **:n** and **:p**  
25208 commands below) shall be re-examined. The *filename* shall be subjected to the process of shell  
25209 word expansions (see Section 2.6 (on page 36)); if more than a single pathname results, the  
25210 effects are unspecified. If *filename* is a number sign (`'#'`), the previously examined file shall be  
25211 re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable file),  
25212 an error message to this effect shall be displayed and the current file and screen shall not change.

## 25213 **Examine Next File**

25214 *Synopsis:*     [*count*] :n

25215 Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If  
25216 *filename* refers to a non-seekable file, the results are unspecified.

## 25217 **Examine Previous File**

25218 *Synopsis:*     [*count*] :p

25219 Examine the previous file. If a number *count* is specified, the *count*th previous file shall be  
25220 examined. If *filename* refers to a non-seekable file, the results are unspecified.

## 25221 **Go to Tag**

25222 *Synopsis:*     :t *tagstring*<newline>

25223 If the file containing the tag named by the *tagstring* argument is not the current file, examine the  
25224 file, as if the **:e** command was executed with that file as the argument. Otherwise, or in addition,  
25225 display the screenful beginning with the tag, as described for the **-t** option (see the OPTIONS  
25226 section). If the *ctags* utility is not supported by the system, the use of **:t** produces undefined  
25227 results.

## 25228 **Invoke Editor**

25229 *Synopsis:*     v

25230 Invoke an editor to edit the current file being examined. If standard input is being examined, the  
25231 results are unspecified. The name of the editor shall be taken from the environment variable  
25232 *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the  
25233 editor shall be invoked with a **-c** *linenumber* command line argument, where *linenumber* is the  
25234 line number of the file line containing the display line currently displayed as the first line of the  
25235 screen. It is implementation-defined whether line-setting options are passed to editors other  
25236 than *vi* and *ex*.

25237 When the editor exits, *more* shall resume with the same file and screen as when the editor was  
25238 invoked.

**Display Position**

Synopsis:       =

                  <control>-G

Write a message for which the information references the first byte of the line after the last line of the file on the screen. This message shall include the name of the file currently being examined, its number relative to the total number of files there are to examine, the line number in the file, the byte number and the total bytes in the file, and what percentage of the file precedes the current position. If *more* is reading from standard input, or the file is shorter than a single screen, the line number, the byte number, the total bytes, and the percentage need not be written.

**Quit**

Synopsis:       q

                  :q

                  ZZ

Exit *more*.

**EXIT STATUS**

The following exit values shall be returned:

- 0   Successful completion.
- >0   An error occurred.

**CONSEQUENCES OF ERRORS**

If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to examine the next file in the argument list, but the final exit status shall be affected. If an error is encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file in the argument list, but the final exit status shall be affected. If an error is encountered accessing a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not be affected.

**APPLICATION USAGE**

When the standard output is not a terminal, only the **-s** filter-modification option is effective. This is based on historical practice. For example, a typical implementation of *man* pipes its output through *more -s* to squeeze excess white space for terminal users. When *man* is piped to *lp*, however, it is undesirable for this squeezing to happen.

**EXAMPLES**

The **-p** allows arbitrary commands to be executed at the start of each file. Examples are:

*more -p G file1 file2*

Examine each file starting with its last screenful.

*more -p 100 file1 file2*

Examine each file starting with line 100 in the current position (usually the third line, so line 98 would be the first line written).

*more -p /100 file1 file2*

Examine each file starting with the first line containing the string "100" in the current position

**RATIONALE**

The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the POSIX file display program since it is more widely available than either the public-domain program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the



25283 features selected; it is almost fully upwards-compatible from the 4.3 BSD version in wide use  
 25284 and has become more amenable for *vi* users. Several features originally derived from various file  
 25285 editors, found in both *less* and *pg*, have been added to this volume of IEEE Std 1003.1-2001 as  
 25286 they have proved extremely popular with users.

25287 There are inconsistencies between *more* and *vi* that result from historical practice. For example,  
 25288 the single-character commands **h**, **f**, **b**, and <space> are screen movers in *more*, but cursor  
 25289 movers in *vi*. These inconsistencies were maintained because the cursor movements are not  
 25290 applicable to *more* and the powerful functionality achieved without the use of the control key  
 25291 justifies the differences.

25292 The tags interface has been included in a program that is not a text editor because it promotes  
 25293 another degree of consistent operation with *vi*. It is conceivable that the paging environment of  
 25294 *more* would be superior for browsing source code files in some circumstances.

25295 The operating mode referred to for block-mode terminals effectively adds a <newline> to each  
 25296 Synopsis line that currently has none. So, for example, **d**<newline> would page one screenful.  
 25297 The mode could be triggered by a command line option, environment variable, or some other  
 25298 method. The details are not imposed by this volume of IEEE Std 1003.1-2001 because there are so  
 25299 few systems known to support such terminals. Nevertheless, it was considered that all systems  
 25300 should be able to support *more* given the exception cited for this small community of terminals  
 25301 because, in comparison to *vi*, the cursor movements are few and the command set relatively  
 25302 amenable to the optional <newline>s.

25303 Some versions of *more* provide a shell escaping mechanism similar to the *ex* ! command. The  
 25304 standard developers did not consider that this was necessary in a paginator, particularly given  
 25305 the wide acceptance of multiple window terminals and job control features. (They chose to  
 25306 retain such features in the editors and *mailx* because the shell interaction also gives an  
 25307 opportunity to modify the editing buffer, which is not applicable to *more*.)

25308 The **-p** (position) option replaces the **+** command because of the Utility Syntax Guidelines. In  
 25309 early proposals, it took a *pattern* argument, but historical *less* provided the *more* general facility of  
 25310 a command. It would have been desirable to use the same **-c** as *ex* and *vi*, but the letter was  
 25311 already in use.

25312 The text stating “from a non-rewindable stream ... implementations may limit the amount of  
 25313 backwards motion supported” would allow an implementation that permitted no backwards  
 25314 motion beyond text already on the screen. It was not possible to require a minimum amount of  
 25315 backwards motion that would be effective for all conceivable device types. The implementation  
 25316 should allow the user to back up as far as possible, within device and reasonable memory  
 25317 allocation constraints.

25318 Historically, non-printable characters were displayed using the ARPA standard mappings,  
 25319 which are as follows:

- 25320 1. Printable characters are left alone.
- 25321 2. Control characters less than \177 are represented as followed by the character offset from  
 25322 the '@' character in the ASCII map; for example, \007 is represented as 'G'.
- 25323 3. \177 is represented as followed by ' ? '.

25324 The display of characters having their eighth bit set was less standard. Existing implementations  
 25325 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed characters with their  
 25326 eighth bit set as the two characters "M-", followed by the seven-bit display as described  
 25327 previously.) The latter probably has the best claim to historical practice because it was used with  
 25328 the **-v** option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

25329 No specific display format is required by IEEE Std 1003.1-2001. Implementations are encouraged  
25330 to conform to historic practice in the absence of any strong reason to diverge.

25331 **FUTURE DIRECTIONS**

25332 None.

25333 **SEE ALSO**

25334 Chapter 2 (on page 29), *ctags*, *ed*, *ex*, *vi*

25335 **CHANGE HISTORY**

25336 First released in Issue 4.

25337 **Issue 5**

25338 The FUTURE DIRECTIONS section is added.

25339 **Issue 6**

25340 This utility is marked as part of the User Portability Utilities option.

25341 The obsolescent SYNOPSIS is removed.

25342 The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:

- 25343 • Changes have been made as a result of IEEE PASC Interpretations 1003.2 #37 and #109.
- 25344 • The *more* utility should be able to handle underlined and emboldened displays of characters
- 25345 that are wider than a single column position.

25346 **NAME**

25347 mv — move files

25348 **SYNOPSIS**25349 mv [-fi] *source\_file target\_file*25350 mv [-fi] *source\_file... target\_file*25351 **DESCRIPTION**

25352 In the first synopsis form, the *mv* utility shall move the file named by the *source\_file* operand to  
 25353 the destination specified by the *target\_file*. This first synopsis form is assumed when the final  
 25354 operand does not name an existing directory and is not a symbolic link referring to an existing  
 25355 directory.

25356 In the second synopsis form, *mv* shall move each file named by a *source\_file* operand to a  
 25357 destination file in the existing directory named by the *target\_dir* operand, or referenced if  
 25358 *target\_dir* is a symbolic link referring to an existing directory. The destination path for each  
 25359 *source\_file* shall be the concatenation of the target directory, a single slash character, and the last  
 25360 pathname component of the *source\_file*. This second form is assumed when the final operand  
 25361 names an existing directory.

25362 If any operand specifies an existing file of a type not specified by the System Interfaces volume  
 25363 of IEEE Std 1003.1-2001, the behavior is implementation-defined.

25364 For each *source\_file* the following steps shall be taken:

25365 1. If the destination path exists, the **-f** option is not specified, and either of the following  
 25366 conditions is true:

25367 a. The permissions of the destination path do not permit writing and the standard input  
 25368 is a terminal.

25369 b. The **-i** option is specified.

25370 the *mv* utility shall write a prompt to standard error and read a line from standard input. If  
 25371 the response is not affirmative, *mv* shall do nothing more with the current *source\_file* and  
 25372 go on to any remaining *source\_files*.

25373 2. The *mv* utility shall perform actions equivalent to the *rename()* function defined in the  
 25374 System Interfaces volume of IEEE Std 1003.1-2001, called with the following arguments:

25375 a. The *source\_file* operand is used as the *old* argument.

25376 b. The destination path is used as the *new* argument.

25377 If this succeeds, *mv* shall do nothing more with the current *source\_file* and go on to any  
 25378 remaining *source\_files*. If this fails for any reasons other than those described for the *errno*  
 25379 [EXDEV] in the System Interfaces volume of IEEE Std 1003.1-2001, *mv* shall write a  
 25380 diagnostic message to standard error, do nothing more with the current *source\_file*, and go  
 25381 on to any remaining *source\_files*.

25382 3. If the destination path exists, and it is a file of type directory and *source\_file* is not a file of  
 25383 type directory, or it is a file not of type directory and *source\_file* is a file of type directory,  
 25384 *mv* shall write a diagnostic message to standard error, do nothing more with the current  
 25385 *source\_file*, and go on to any remaining *source\_files*.

25386 4. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*  
 25387 shall write a diagnostic message to standard error, do nothing more with the current  
 25388 *source\_file*, and go on to any remaining *source\_files*.

- 25389 5. The file hierarchy rooted in *source\_file* shall be duplicated as a file hierarchy rooted in the  
 25390 destination path. If *source\_file* or any of the files below it in the hierarchy are symbolic  
 25391 links, the links themselves shall be duplicated, including their contents, rather than any  
 25392 files to which they refer. The following characteristics of each file in the file hierarchy shall  
 25393 be duplicated:
- 25394 • The time of last data modification and time of last access
  - 25395 • The user ID and group ID
  - 25396 • The file mode
- 25397 If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode  
 25398 bits S\_ISUID and S\_ISGID shall not be duplicated.
- 25399 When files are duplicated to another file system, the implementation may require that the  
 25400 process invoking *mv* has read access to each file being duplicated.
- 25401 If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic  
 25402 message to standard error, do nothing more with the current *source\_file*, and go on to any  
 25403 remaining *source\_files*.
- 25404 If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic  
 25405 message to standard error, but this failure shall not cause *mv* to modify its exit status.
- 25406 6. The file hierarchy rooted in *source\_file* shall be removed. If this fails for any reason, *mv* shall  
 25407 write a diagnostic message to the standard error, do nothing more with the current  
 25408 *source\_file*, and go on to any remaining *source\_files*.

#### 25409 OPTIONS

- 25410 The *mv* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 25411 Utility Syntax Guidelines.
- 25412 The following options shall be supported:
- 25413 **-f** Do not prompt for confirmation if the destination path exists. Any previous  
 25414 occurrence of the **-i** option is ignored.
  - 25415 **-i** Prompt for confirmation if the destination path exists. Any previous occurrence of  
 25416 the **-f** option is ignored.
- 25417 Specifying more than one of the **-f** or **-i** options shall not be considered an error. The last option  
 25418 specified shall determine the behavior of *mv*.

#### 25419 OPERANDS

- 25420 The following operands shall be supported:
- 25421 *source\_file* A pathname of a file or directory to be moved.
  - 25422 *target\_file* A new pathname for the file or directory being moved.
  - 25423 *target\_dir* A pathname of an existing directory into which to move the input files.

#### 25424 STDIN

- 25425 The standard input shall be used to read an input line in response to each prompt specified in  
 25426 the STDERR section. Otherwise, the standard input shall not be used.

#### 25427 INPUT FILES

- 25428 The input files specified by each *source\_file* operand can be of any file type.

25429 **ENVIRONMENT VARIABLES**

25430 The following environment variables shall affect the execution of *mv*:

25431 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 25432 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 25433 Internationalization Variables for the precedence of internationalization variables  
 25434 used to determine the values of locale categories.)

25435 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 25436 internationalization variables.

25437 **LC\_COLLATE**  
 25438 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 25439 character collating elements used in the extended regular expression defined for  
 25440 the **yesexpr** locale keyword in the **LC\_MESSAGES** category.

25441 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 25442 characters (for example, single-byte as opposed to multi-byte characters in  
 25443 arguments and input files), the behavior of character classes used in the extended  
 25444 regular expression defined for the **yesexpr** locale keyword in the **LC\_MESSAGES**  
 25445 category.

25446 **LC\_MESSAGES**  
 25447 Determine the locale for the processing of affirmative responses that should be  
 25448 used to affect the format and contents of diagnostic messages written to standard  
 25449 error.

25450 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

25451 **ASYNCHRONOUS EVENTS**

25452 Default.

25453 **STDOUT**

25454 Not used.

25455 **STDERR**

25456 Prompts shall be written to the standard error under the conditions specified in the  
 25457 **DESCRIPTION** section. The prompts shall contain the destination pathname, but their format is  
 25458 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

25459 **OUTPUT FILES**

25460 The output files may be of any file type.

25461 **EXTENDED DESCRIPTION**

25462 None.

25463 **EXIT STATUS**

25464 The following exit values shall be returned:

25465 0 All input files were moved successfully.

25466 >0 An error occurred.

25467 **CONSEQUENCES OF ERRORS**

25468 If the copying or removal of *source\_file* is prematurely terminated by a signal or error, *mv* may  
 25469 leave a partial copy of *source\_file* at the source or destination. The *mv* utility shall not modify  
 25470 both *source\_file* and the destination path simultaneously; termination at any point shall leave  
 25471 either *source\_file* or the destination path complete.

25472 **APPLICATION USAGE**

25473 Some implementations mark for update the *st\_ctime* field of renamed files and some do not.  
 25474 Applications which make use of the *st\_ctime* field may behave differently with respect to  
 25475 renamed files unless they are designed to allow for either behavior.

25476 **EXAMPLES**

25477 If the current directory contains only files **a** (of any type defined by the System Interfaces  
 25478 volume of IEEE Std 1003.1-2001), **b** (also of any type), and a directory **c**:

25479 `mv a b c`  
 25480 `mv c d`

25481 results with the original files **a** and **b** residing in the directory **d** in the current directory.

25482 **RATIONALE**

25483 Early proposals diverged from the SVID and BSD historical practice in that they required that  
 25484 when the destination path exists, the **-f** option is not specified, and input is not a terminal, *mv*  
 25485 fails. This was done for compatibility with *cp*. The current text returns to historical practice. It  
 25486 should be noted that this is consistent with the *rename()* function defined in the System  
 25487 Interfaces volume of IEEE Std 1003.1-2001, which does not require write permission on the  
 25488 target.

25489 For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for  
 25490 confirmation, should be interpreted in the following manner:

25491 `if (exists AND (NOT f_option) AND`  
 25492 `((not_writable AND input_is_terminal) OR i_option))`

25493 The **-i** option exists on BSD systems, giving applications and users a way to avoid accidentally  
 25494 unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv*  
 25495 deletes all existing destination paths without prompting, even when **-i** is specified; this is  
 25496 inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when  
 25497 the file is unwritable and the standard input is not a terminal. The standard developers decided  
 25498 that use of **-i** is a request for interaction, so when the destination path exists, the utility takes  
 25499 instructions from whatever responds to standard input.

25500 The *rename()* function is able to move directories within the same file system. Some historical  
 25501 versions of *mv* have been able to move directories, but not to a different file system. The  
 25502 standard developers considered that this was an annoying inconsistency, so this volume of  
 25503 IEEE Std 1003.1-2001 requires directories to be able to be moved even across file systems. There  
 25504 is no **-R** option to confirm that moving a directory is actually intended, since such an option was  
 25505 not required for moving directories in historical practice. Requiring the application to specify it  
 25506 sometimes, depending on the destination, seemed just as inconsistent. The semantics of the  
 25507 *rename()* function were preserved as much as possible. For example, *mv* is not permitted to  
 25508 “rename” files to or from directories, even though they might be empty and removable.

25509 Historic implementations of *mv* did not exit with a non-zero exit status if they were unable to  
 25510 duplicate any file characteristics when moving a file across file systems, nor did they write a  
 25511 diagnostic message for the user. The former behavior has been preserved to prevent scripts from  
 25512 breaking; a diagnostic message is now required, however, so that users are alerted that the file  
 25513 characteristics have changed.

25514 The exact format of the interactive prompts is unspecified. Only the general nature of the  
 25515 contents of prompts are specified because implementations may desire more descriptive  
 25516 prompts than those used on historical implementations. Therefore, an application not using the  
 25517 **-f** option or using the **-i** option relies on the system to provide the most suitable dialog directly  
 25518 with the user, based on the behavior specified.

25519 When *mv* is dealing with a single file system and *source\_file* is a symbolic link, the link itself is  
25520 moved as a consequence of the dependence on the *rename()* functionality, per the  
25521 DESCRIPTION. Across file systems, this has to be made explicit.

25522 **FUTURE DIRECTIONS**

25523 None.

25524 **SEE ALSO**

25525 *cp*, *ln*, the System Interfaces volume of IEEE Std 1003.1-2001, *rename()*

25526 **CHANGE HISTORY**

25527 First released in Issue 2.

25528 **Issue 6**

25529 The *mv* utility is changed to describe processing of symbolic links as specified in the  
25530 IEEE P1003.2b draft standard.

25531 The APPLICATION USAGE section is added.

25532 **NAME**

25533 newgrp — change to a new group

25534 **SYNOPSIS**

25535 UP newgrp [-l] [group]

25536

25537 **DESCRIPTION**

25538 The *newgrp* utility shall create a new shell execution environment with a new real and effective  
 25539 group identification. Of the attributes listed in Section 2.12 (on page 61), the new shell execution  
 25540 environment shall retain the working directory, file creation mask, and exported variables from  
 25541 the previous environment (that is, open files, traps, unexported variables, alias definitions, shell  
 25542 functions, and *set* options may be lost). All other aspects of the process environment that are  
 25543 preserved by the *exec* family of functions defined in the System Interfaces volume of  
 25544 IEEE Std 1003.1-2001 shall also be preserved by *newgrp*; whether other aspects are preserved is  
 25545 unspecified.

25546 A failure to assign the new group identifications (for example, for security or password-related  
 25547 reasons) shall not prevent the new shell execution environment from being created.

25548 The *newgrp* utility shall affect the supplemental groups for the process as follows:

- 25549 • On systems where the effective group ID is normally in the supplementary group list (or  
 25550 whenever the old effective group ID actually is in the supplementary group list):
  - 25551 — If the new effective group ID is also in the supplementary group list, *newgrp* shall change  
 25552 the effective group ID.
  - 25553 — If the new effective group ID is not in the supplementary group list, *newgrp* shall add the  
 25554 new effective group ID to the list, if there is room to add it.
- 25555 • On systems where the effective group ID is not normally in the supplementary group list (or  
 25556 whenever the old effective group ID is not in the supplementary group list):
  - 25557 — If the new effective group ID is in the supplementary group list, *newgrp* shall delete it.
  - 25558 — If the old effective group ID is not in the supplementary list, *newgrp* shall add it if there is  
 25559 room.

25560 **Note:** The System Interfaces volume of IEEE Std 1003.1-2001 does not specify whether the effective  
 25561 group ID of a process is included in its supplementary group list.

25562 With no operands, *newgrp* shall change the effective group back to the groups identified in the  
 25563 user's user entry, and shall set the list of supplementary groups to that set in the user's group  
 25564 database entries.

25565 If a password is required for the specified group, and the user is not listed as a member of that  
 25566 group in the group database, the user shall be prompted to enter the correct password for that  
 25567 group. If the user is listed as a member of that group, no password shall be requested. If no  
 25568 password is required for the specified group, it is implementation-defined whether users not  
 25569 listed as members of that group can change to that group. Whether or not a password is  
 25570 required, implementation-defined system accounting or security mechanisms may impose  
 25571 additional authorization restrictions that may cause *newgrp* to write a diagnostic message and  
 25572 suppress the changing of the group identification.

25573 **OPTIONS**

25574 The *newgrp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 25575 12.2, Utility Syntax Guidelines.



25576 The following option shall be supported:

25577 **-l** (The letter ell.) Change the environment to what would be expected if the user  
25578 actually logged in again.

#### 25579 **OPERANDS**

25580 The following operand shall be supported:

25581 *group* A group name from the group database or a non-negative numeric group ID.  
25582 Specifies the group ID to which the real and effective group IDs shall be set. If  
25583 *group* is a non-negative numeric string and exists in the group database as a group  
25584 name (see *getgrnam()*), the numeric group ID associated with that group name  
25585 shall be used as the group ID.

#### 25586 **STDIN**

25587 Not used.

#### 25588 **INPUT FILES**

25589 The file */dev/tty* shall be used to read a single line of text for password checking, when one is  
25590 required.

#### 25591 **ENVIRONMENT VARIABLES**

25592 The following environment variables shall affect the execution of *newgrp*:

25593 *LANG* Provide a default value for the internationalization variables that are unset or null.  
25594 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
25595 Internationalization Variables for the precedence of internationalization variables  
25596 used to determine the values of locale categories.)

25597 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
25598 internationalization variables.

25599 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
25600 characters (for example, single-byte as opposed to multi-byte characters in  
25601 arguments).

25602 *LC\_MESSAGES*

25603 Determine the locale that should be used to affect the format and contents of  
25604 diagnostic messages written to standard error.

25605 *XS* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 25606 **ASYNCHRONOUS EVENTS**

25607 Default.

#### 25608 **STDOUT**

25609 Not used.

#### 25610 **STDERR**

25611 The standard error shall be used for diagnostic messages and a prompt string for a password, if  
25612 one is required. Diagnostic messages may be written in cases where the exit status is not  
25613 available. See the EXIT STATUS section.

#### 25614 **OUTPUT FILES**

25615 None.

#### 25616 **EXTENDED DESCRIPTION**

25617 None.

25618 **EXIT STATUS**

25619 If *newgrp* succeeds in creating a new shell execution environment, whether or not the group  
25620 identification was changed successfully, the exit status shall be the exit status of the shell.  
25621 Otherwise, the following exit value shall be returned:

25622 >0 An error occurred.

25623 **CONSEQUENCES OF ERRORS**

25624 The invoking shell may terminate.

25625 **APPLICATION USAGE**

25626 There is no convenient way to enter a password into the group database. Use of group  
25627 passwords is not encouraged, because by their very nature they encourage poor security  
25628 practices. Group passwords may disappear in the future.

25629 A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with  
25630 *newgrp*, which in turn overlays itself with a new shell after changing group. On some  
25631 implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

25632 The *newgrp* command is intended only for use from an interactive terminal. It does not offer a  
25633 useful interface for the support of applications.

25634 The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it  
25635 successfully invokes a new shell and the rest of the original shell script is bypassed when the  
25636 new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems.  
25637 But usage such as:

25638 newgrp foo  
25639 echo \$?

25640 is not useful because the new shell might not have access to any status *newgrp* may have  
25641 generated (and most historical systems do not provide this status). A zero status echoed here  
25642 does not necessarily indicate that the user has changed to the new group successfully. Following  
25643 *newgrp* with the *id* command provides a portable means of determining whether the group  
25644 change was successful or not.

25645 **EXAMPLES**

25646 None.

25647 **RATIONALE**

25648 Most historical implementations use one of the *exec* functions to implement the behavior of  
25649 *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected  
25650 after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp*  
25651 issue a diagnostic message to tell the user that the environment changed, it would be  
25652 inappropriate to require this change to some historical implementations.

25653 The password mechanism is allowed in the group database, but how this would be  
25654 implemented is not specified.

25655 The *newgrp* utility was retained in this volume of IEEE Std 1003.1-2001, even given the existence  
25656 of the multiple group permissions feature in the System Interfaces volume of  
25657 IEEE Std 1003.1-2001, for several reasons. First, in some implementations, the group ownership  
25658 of a newly created file is determined by the group of the directory in which the file is created, as  
25659 allowed by the System Interfaces volume of IEEE Std 1003.1-2001; on other implementations, the  
25660 group ownership of a newly created file is determined by the effective group ID. On  
25661 implementations of the latter type, *newgrp* allows files to be created with a specific group  
25662 ownership. Finally, many implementations use the real group ID in accounting, and on such  
25663 systems, *newgrp* allows the accounting identity of the user to be changed.

25664 **FUTURE DIRECTIONS**

25665           None.

25666 **SEE ALSO**

25667           Chapter 2 (on page 29), *sh*, the System Interfaces volume of IEEE Std 1003.1-2001, *exec*,  
25668           *getgrnam()*

25669 **CHANGE HISTORY**

25670           First released in Issue 2.

25671 **Issue 6**

25672           This utility is marked as part of the User Portability Utilities option.

25673           The obsolescent SYNOPSIS is removed.

25674           The text describing supplemental groups is no longer conditional on {NGROUPS\_MAX} being  
25675           greater than 1. This is because {NGROUPS\_MAX} now has a minimum value of 8. This is a FIPS  
25676           requirement.

25677 **NAME**

25678       *nice* — invoke a utility with an altered nice value

25679 **SYNOPSIS**

25680 UP       *nice* [-n *increment*] *utility* [*argument...*]

25681

25682 **DESCRIPTION**

25683       The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see the  
25684       Base Definitions volume of IEEE Std 1003.1-2001, Section 3.239, Nice Value). With no options  
25685       and only if the user has appropriate privileges, the executed utility shall be run with a nice value  
25686       that is some implementation-defined quantity less than or equal to the nice value of the current  
25687       process. If the user lacks appropriate privileges to affect the nice value in the requested manner,  
25688       the *nice* utility shall not affect the nice value; in this case, a warning message may be written to  
25689       standard error, but this shall not prevent the invocation of *utility* or affect the exit status.

25690 **OPTIONS**

25691       The *nice* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
25692       12.2, Utility Syntax Guidelines.

25693       The following option is supported:

25694       -n *increment*   A positive or negative decimal integer which shall have the same effect on the  
25695       execution of the utility as if the utility had called the *nice*() function with the  
25696       numeric value of the *increment* option-argument.

25697 **OPERANDS**

25698       The following operands shall be supported:

25699       *utility*        The name of a utility that is to be invoked. If the *utility* operand names any of the  
25700       special built-in utilities in Section 2.14 (on page 64), the results are undefined.

25701       *argument*     Any string to be supplied as an argument when invoking the utility named by the  
25702       *utility* operand.

25703 **STDIN**

25704       Not used.

25705 **INPUT FILES**

25706       None.

25707 **ENVIRONMENT VARIABLES**

25708       The following environment variables shall affect the execution of *nice*:

25709       *LANG*        Provide a default value for the internationalization variables that are unset or null.  
25710       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
25711       Internationalization Variables for the precedence of internationalization variables  
25712       used to determine the values of locale categories.)

25713       *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
25714       internationalization variables.

25715       *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
25716       characters (for example, single-byte as opposed to multi-byte characters in  
25717       arguments).

25718       *LC\_MESSAGES*

25719       Determine the locale that should be used to affect the format and contents of  
25720       diagnostic messages written to standard error.

|       |     |                                                 |                                                                                                                                                     |
|-------|-----|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| 25721 | XSI | <b>NLSPATH</b>                                  | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                               |
| 25722 |     | <b>PATH</b>                                     | Determine the search path used to locate the utility to be invoked. See the Base                                                                    |
| 25723 |     |                                                 | Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.                                                                       |
| 25724 |     | <b>ASYNCHRONOUS EVENTS</b>                      |                                                                                                                                                     |
| 25725 |     |                                                 | Default.                                                                                                                                            |
| 25726 |     | <b>STDOUT</b>                                   |                                                                                                                                                     |
| 25727 |     |                                                 | Not used.                                                                                                                                           |
| 25728 |     | <b>STDERR</b>                                   |                                                                                                                                                     |
| 25729 |     |                                                 | The standard error shall be used only for diagnostic messages.                                                                                      |
| 25730 |     | <b>OUTPUT FILES</b>                             |                                                                                                                                                     |
| 25731 |     |                                                 | None.                                                                                                                                               |
| 25732 |     | <b>EXTENDED DESCRIPTION</b>                     |                                                                                                                                                     |
| 25733 |     |                                                 | None.                                                                                                                                               |
| 25734 |     | <b>EXIT STATUS</b>                              |                                                                                                                                                     |
| 25735 |     |                                                 | If <i>utility</i> is invoked, the exit status of <i>nice</i> shall be the exit status of <i>utility</i> ; otherwise, the <i>nice</i>                |
| 25736 |     |                                                 | utility shall exit with one of the following values:                                                                                                |
| 25737 |     | 1-125                                           | An error occurred in the <i>nice</i> utility.                                                                                                       |
| 25738 |     | 126                                             | The utility specified by <i>utility</i> was found but could not be invoked.                                                                         |
| 25739 |     | 127                                             | The utility specified by <i>utility</i> could not be found.                                                                                         |
| 25740 |     | <b>CONSEQUENCES OF ERRORS</b>                   |                                                                                                                                                     |
| 25741 |     |                                                 | Default.                                                                                                                                            |
| 25742 |     | <b>APPLICATION USAGE</b>                        |                                                                                                                                                     |
| 25743 |     |                                                 | The only guaranteed portable uses of this utility are:                                                                                              |
| 25744 |     | <i>nice utility</i>                             |                                                                                                                                                     |
| 25745 |     |                                                 | Run <i>utility</i> with the default lower nice value.                                                                                               |
| 25746 |     | <i>nice -n &lt;positive integer&gt; utility</i> |                                                                                                                                                     |
| 25747 |     |                                                 | Run <i>utility</i> with a lower nice value.                                                                                                         |
| 25748 |     |                                                 | On some implementations they have no discernible effect on the invoked utility and on some                                                          |
| 25749 |     |                                                 | others they are exactly equivalent.                                                                                                                 |
| 25750 |     |                                                 | Historical systems have frequently supported the <i>&lt;positive integer&gt;</i> up to 20. Since there is no                                        |
| 25751 |     |                                                 | error penalty associated with guessing a number that is too high, users without access to the                                                       |
| 25752 |     |                                                 | system conformance document (to see what limits are actually in place) could use the historical                                                     |
| 25753 |     |                                                 | 1 to 20 range or attempt to use very large numbers if the job should be truly low priority.                                                         |
| 25754 |     |                                                 | The nice value of a process can be displayed using the command:                                                                                     |
| 25755 |     | <i>ps -o nice</i>                               |                                                                                                                                                     |
| 25756 |     |                                                 | The <i>command</i> , <i>env</i> , <i>nice</i> , <i>nohup</i> , <i>time</i> , and <i>xargs</i> utilities have been specified to use exit code 127 if |
| 25757 |     |                                                 | an error occurs so that applications can distinguish “failure to find a utility” from “invoked                                                      |
| 25758 |     |                                                 | utility exited with an error indication”. The value 127 was chosen because it is not commonly                                                       |
| 25759 |     |                                                 | used for other meanings; most utilities use small values for “normal error conditions” and the                                                      |
| 25760 |     |                                                 | values above 128 can be confused with termination due to receipt of a signal. The value 126 was                                                     |
| 25761 |     |                                                 | chosen in a similar manner to indicate that the utility could be found, but not invoked. Some                                                       |
| 25762 |     |                                                 | scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction                                                    |
| 25763 |     |                                                 | between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to                                                    |

25764 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
25765 any other reason.

#### 25766 **EXAMPLES**

25767 None.

#### 25768 **RATIONALE**

25769 Due to the text about the limits of the *nice* value being implementation-defined, *nice* is not  
25770 actually required to change the *nice* value of the executed command; the limits could be zero  
25771 differences from the system default, although the implementor is required to document this fact  
25772 in the conformance document.

25773 The 4.3 BSD version of *nice* does not check whether *increment* is a valid decimal integer. The  
25774 command *nice -x utility*, for example, would be treated the same as the command *nice --1*  
25775 *utility*. If the user does not have appropriate privileges, this results in a “permission denied”  
25776 error. This is considered a bug.

25777 When a user without appropriate privileges gives a negative *increment*, System V treats it like  
25778 the command *nice -0 utility*, while 4.3 BSD writes a “permission denied” message and does not  
25779 run the utility. Neither was considered clearly superior, so the behavior was left unspecified.

25780 The C shell has a built-in version of *nice* that has a different interface from the one described in  
25781 this volume of IEEE Std 1003.1-2001.

25782 The term “utility” is used, rather than “command”, to highlight the fact that shell compound  
25783 commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used.  
25784 However, “utility” includes user application programs and shell scripts, not just utilities defined  
25785 in this volume of IEEE Std 1003.1-2001.

25786 Historical implementations of *nice* provide a *nice* value range of 40 or 41 discrete steps, with the  
25787 default *nice* value being the midpoint of that range. By default, they lower the *nice* value of the  
25788 executed utility by 10.

25789 Some historical documentation states that the *increment* value must be within a fixed range. This  
25790 is misleading; the valid *increment* values on any invocation are determined by the current  
25791 process *nice* value, which is not always the default.

25792 The definition of *nice* value is not intended to suggest that all processes in a system have  
25793 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the  
25794 System Interfaces volume of IEEE Std 1003.1-2001 make the notion of a single underlying  
25795 priority for all scheduling policies problematic. Some implementations may implement the *nice*-  
25796 related features to affect all processes on the system, others to affect just the general time-  
25797 sharing activities implied by this volume of IEEE Std 1003.1-2001, and others may have no effect  
25798 at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of  
25799 implementation strategies are possible.

#### 25800 **FUTURE DIRECTIONS**

25801 None.

#### 25802 **SEE ALSO**

25803 Chapter 2 (on page 29), *renice*, the System Interfaces volume of IEEE Std 1003.1-2001, *nice()*

#### 25804 **CHANGE HISTORY**

25805 First released in Issue 4.

25806 **Issue 6**

25807 This utility is marked as part of the User Portability Utilities option.

25808 The obsolescent SYNOPSIS is removed.

25809 **NAME**

25810 nl — line numbering filter

25811 **SYNOPSIS**

```
25812 xsi nl [-p] [-b type] [-d delim] [-f type] [-h type] [-i incr] [-l num] [-n format]
25813 [-s sep] [-v startnum] [-w width] [file]
```

25814

25815 **DESCRIPTION**

25816 The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and  
 25817 shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional  
 25818 functionality may be provided in accordance with the command options in effect.

25819 The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at  
 25820 the start of each logical page. A logical page consists of a header, a body, and a footer section.  
 25821 Empty sections are valid. Different line numbering options are independently available for  
 25822 header, body, and footer (for example, no numbering of header and footer lines while  
 25823 numbering blank lines only in the body).

25824 The starts of logical page sections shall be signaled by input lines containing nothing but the  
 25825 following delimiter characters:

25826

25827

25828

25829

| Line     | Start of |
|----------|----------|
| \: \: \: | Header   |
| \: \:    | Body     |
| \:       | Footer   |

25830 Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

25831 **OPTIONS**

25832 The *nl* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 25833 Utility Syntax Guidelines. Only one file can be named.

25834 The following options shall be supported:

25835 **-b type** Specify which logical page body lines shall be numbered. Recognized *types* and  
 25836 their meaning are:

25837 **a** Number all lines.

25838 **t** Number only non-empty lines.

25839 **n** No line numbering.

25840 **pstring** Number only lines that contain the basic regular expression specified in  
 25841 *string*.

25842 The default *type* for logical page body shall be **t** (text lines numbered).

25843 **-d delim** Specify the delimiter characters that indicate the start of a logical page section.  
 25844 These can be changed from the default characters "\:" to two user-specified  
 25845 characters. If only one character is entered, the second character shall remain the  
 25846 default character ' : '.

25847 **-f type** Specify the same as **b type** except for footer. The default for logical page footer  
 25848 shall be **n** (no lines numbered).

25849 **-h type** Specify the same as **b type** except for header. The default *type* for logical page  
 25850 header shall be **n** (no lines numbered).



|       |                    |                                                                                                                                                                                                                                                                                         |
|-------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 25851 | <b>-i incr</b>     | Specify the increment value used to number logical page lines. The default shall be 1.                                                                                                                                                                                                  |
| 25852 |                    |                                                                                                                                                                                                                                                                                         |
| 25853 | <b>-l num</b>      | Specify the number of blank lines to be considered as one. For example, <b>-l 2</b> results in only the second adjacent blank line being numbered (if the appropriate <b>-h a</b> , <b>-b a</b> , or <b>-f a</b> option is set). The default shall be 1.                                |
| 25854 |                    |                                                                                                                                                                                                                                                                                         |
| 25855 |                    |                                                                                                                                                                                                                                                                                         |
| 25856 | <b>-n format</b>   | Specify the line numbering format. Recognized values are: <b>ln</b> , left justified, leading zeros suppressed; <b>rn</b> , right justified, leading zeros suppressed; <b>rz</b> , right justified, leading zeros kept. The default <i>format</i> shall be <b>rn</b> (right justified). |
| 25857 |                    |                                                                                                                                                                                                                                                                                         |
| 25858 |                    |                                                                                                                                                                                                                                                                                         |
| 25859 | <b>-p</b>          | Specify that numbering should not be restarted at logical page delimiters.                                                                                                                                                                                                              |
| 25860 | <b>-s sep</b>      | Specify the characters used in separating the line number and the corresponding text line. The default <i>sep</i> shall be a <tab>.                                                                                                                                                     |
| 25861 |                    |                                                                                                                                                                                                                                                                                         |
| 25862 | <b>-v startnum</b> | Specify the initial value used to number logical page lines. The default shall be 1.                                                                                                                                                                                                    |
| 25863 | <b>-w width</b>    | Specify the number of characters to be used for the line number. The default <i>width</i> shall be 6.                                                                                                                                                                                   |
| 25864 |                    |                                                                                                                                                                                                                                                                                         |

## 25865 OPERANDS

25866 The following operand shall be supported:

25867 *file* A pathname of a text file to be line-numbered.

## 25868 STDIN

25869 The standard input is a text file that is used if no *file* operand is given.

## 25870 INPUT FILES

25871 The input file named by the *file* operand is a text file.

## 25872 ENVIRONMENT VARIABLES

25873 The following environment variables shall affect the execution of *nl*:

25874 **LANG** Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

25878 **LC\_ALL** If set to a non-empty string value, override the values of all the other internationalization variables.

25880 **LC\_COLLATE** Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

25883 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, and for deciding which characters are in character class **graph** (for the **-b t**, **-f t**, and **-h t** options).

25888 **LC\_MESSAGES** Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

25891 **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

25892 **ASYNCHRONOUS EVENTS**

25893 Default.

25894 **STDOUT**

25895 The standard output shall be a text file in the following format:

25896 "%s%s%s", &lt;line number&gt;, &lt;separator&gt;, &lt;input line&gt;

25897 where &lt;line number&gt; is one of the following numeric formats:

25898 %6d When the **rn** format is used (the default; see **-n**).25899 %06d When the **rz** format is used.25900 %-6d When the **ln** format is used.25901 <empty> When line numbers are suppressed for a portion of the page; the <separator> is also  
25902 suppressed.25903 In the preceding list, the number 6 is the default width; the **-w** option can change this value.25904 **STDERR**

25905 The standard error shall be used only for diagnostic messages.

25906 **OUTPUT FILES**

25907 None.

25908 **EXTENDED DESCRIPTION**

25909 None.

25910 **EXIT STATUS**

25911 The following exit values shall be returned:

25912 0 Successful completion.

25913 &gt;0 An error occurred.

25914 **CONSEQUENCES OF ERRORS**

25915 Default.

25916 **APPLICATION USAGE**25917 In using the **-d delim** option, care should be taken to escape characters that have special meaning  
25918 to the command interpreter.25919 **EXAMPLES**

25920 The command:

25921 nl -v 10 -i 10 -d \!+ file1

25922 numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is  
25923 " !+ ". Note that the ' ! ' has to be escaped when using *csh* as a command interpreter because of  
25924 its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but does not do any  
25925 harm.25926 **RATIONALE**

25927 None.

25928 **FUTURE DIRECTIONS**

25929 None.

25930 **SEE ALSO**25931 *pr*25932 **CHANGE HISTORY**

25933 First released in Issue 2.

25934 **Issue 5**

25935 The option [**-f type**] is added to the SYNOPSIS. The option descriptions are presented in  
25936 alphabetic order. The description of **-bt** is changed to “Number only non-empty lines”.

25937 **Issue 6**

25938 The obsolescent behavior allowing the options to be intermingled with the optional *file* operand  
25939 is removed.

25940 **NAME**25941 nm — write the name list of an object file (**DEVELOPMENT**)25942 **SYNOPSIS**25943 UP SD XSI nm [-APv] [-efox] [ -g | -u ] [-t *format*] *file*...

25944

25945 **DESCRIPTION**

25946 This utility shall be provided on systems that support both the User Portability Utilities option  
 25947 and the Software Development Utilities option. On other systems it is optional. Certain options  
 25948 are only available on XSI-conformant systems.

25949 The *nm* utility shall display symbolic information appearing in the object file, executable file, or  
 25950 object-file library named by *file*. If no symbolic information is available for a valid input file, the  
 25951 *nm* utility shall report that fact, but not consider it an error condition.

25952 XSI The default base used when numeric values are written is unspecified. On XSI-conformant  
 25953 systems, it shall be decimal.

25954 **OPTIONS**

25955 The *nm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 25956 12.2, Utility Syntax Guidelines.

25957 The following options shall be supported:

25958 **-A** Write the full pathname or library name of an object on each line.

25959 XSI **-e** Write only external (global) and static symbol information.

25960 XSI **-f** Produce full output. Write redundant symbols (**.text**, **.data**, and **.bss**), normally  
 25961 suppressed.

25962 **-g** Write only external (global) symbol information.

25963 XSI **-o** Write numeric values in octal (equivalent to **-t o**).

25964 **-P** Write information in a portable output format, as specified in the STDOUT section.

25965 **-t *format*** Write each numeric value in the specified format. The format shall be dependent  
 25966 on the single character used as the *format* option-argument:

25967 XSI d The offset is written in decimal (default).

25968 o The offset is written in octal.

25969 x The offset is written in hexadecimal.

25970 **-u** Write only undefined symbols.

25971 **-v** Sort output by value instead of alphabetically.

25972 XSI **-x** Write numeric values in hexadecimal (equivalent to **-t x**).

25973 **OPERANDS**

25974 The following operand shall be supported:

25975 *file* A pathname of an object file, executable file, or object-file library.

25976 **STDIN**

25977 See the INPUT FILES section.

25978 **INPUT FILES**

25979       The input file shall be an object file, an object-file library whose format is the same as those  
 25980       produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept  
 25981       additional implementation-defined object library formats for the input file.

25982 **ENVIRONMENT VARIABLES**

25983       The following environment variables shall affect the execution of *nm*:

25984       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 25985       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 25986       Internationalization Variables for the precedence of internationalization variables  
 25987       used to determine the values of locale categories.)

25988       *LC\_ALL*      If set to a non-empty string value, override the values of all the other  
 25989       internationalization variables.

25990       *LC\_COLLATE*   Determine the locale for character collation information for the symbol-name and  
 25991       symbol-value collation sequences.  
 25992

25993       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 25994       characters (for example, single-byte as opposed to multi-byte characters in  
 25995       arguments).

25996       *LC\_MESSAGES*   Determine the locale that should be used to affect the format and contents of  
 25997       diagnostic messages written to standard error.  
 25998

25999 xSI       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

26000 **ASYNCHRONOUS EVENTS**

26001       Default.

26002 **STDOUT**

26003       If symbolic information is present in the input files, then for each file or for each member of an  
 26004       archive, the *nm* utility shall write the following information to standard output. By default, the  
 26005       format is unspecified, but the output shall be sorted alphabetically by symbol name:

- 26006       • Library or object name, if **-A** is specified
- 26007       • Symbol name
- 26008       • Symbol type, which shall either be one of the following single characters or an  
 26009       implementation-defined type represented by a single character:
- 26010       A   Global absolute symbol.
- 26011       a   Local absolute symbol.
- 26012       B   Global “bss” (that is, uninitialized data space) symbol.
- 26013       b   Local bss symbol.
- 26014       D   Global data symbol.
- 26015       d   Local data symbol.
- 26016       T   Global text symbol.
- 26017       t   Local text symbol.
- 26018       U   Undefined symbol.

26019           • Value of the symbol

26020           • The size associated with the symbol, if applicable

26021       This information may be supplemented by additional information specific to the  
26022       implementation.

26023       If the **-P** option is specified, the previous information shall be displayed using the following  
26024       portable format. The three versions differ depending on whether **-t d**, **-t o**, or **-t x** was specified,  
26025       respectively:

26026       "*%s%s %s %d %d\n*", *<library/object name>*, *<name>*, *<type>*,  
26027       *<value>*, *<size>*

26028       "*%s%s %s %o %o\n*", *<library/object name>*, *<name>*, *<type>*,  
26029       *<value>*, *<size>*

26030       "*%s%s %s %x %x\n*", *<library/object name>*, *<name>*, *<type>*,  
26031       *<value>*, *<size>*

26032       where *<library/object name>* shall be formatted as follows:

26033           • If **-A** is not specified, *<library/object name>* shall be an empty string.

26034           • If **-A** is specified and the corresponding *file* operand does not name a library:

26035           *%s: "*, *<file>*

26036           • If **-A** is specified and the corresponding *file* operand names a library. In this case, *<object file>*  
26037           shall name the object file in the library containing the symbol being described:

26038           *%s[%s]: "*, *<file>*, *<object file>*

26039       If **-A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is  
26040       specified and it names a library, *nm* shall write a line identifying the object containing the  
26041       following symbols before the lines containing those symbols, in the form:

26042           • If the corresponding *file* operand does not name a library:

26043           *%s:\n*, *<file>*

26044           • If the corresponding *file* operand names a library; in this case, *<object file>* shall be the name  
26045           of the file in the library containing the following symbols:

26046           *%s[%s]:\n*, *<file>*, *<object file>*

26047       If **-P** is specified, but **-t** is not, the format shall be as if **-t x** had been specified.

26048   **STDERR**

26049       The standard error shall be used only for diagnostic messages.

26050   **OUTPUT FILES**

26051       None.

26052   **EXTENDED DESCRIPTION**

26053       None.

26054   **EXIT STATUS**

26055       The following exit values shall be returned:

26056       0   Successful completion.

26057       >0   An error occurred.

26058 **CONSEQUENCES OF ERRORS**

26059 Default.

26060 **APPLICATION USAGE**

26061 Mechanisms for dynamic linking make this utility less meaningful when applied to an  
26062 executable file because a dynamically linked executable may omit numerous library routines  
26063 that would be found in a statically linked executable.

26064 **EXAMPLES**

26065 None.

26066 **RATIONALE**

26067 Historical implementations of *nm* have used different bases for numeric output and supplied  
26068 different default types of symbols that were reported. The **-t format** option, similar to that used  
26069 in *od* and *strings*, can be used to specify the numeric base; **-g** and **-u** can be used to restrict the  
26070 amount of output or the types of symbols included in the output.

26071 The compromise of using **-t format** versus using **-d**, **-o**, and other similar options was necessary  
26072 because of differences in the meaning of **-o** between implementations. The **-o** option from BSD  
26073 has been provided here as **-A** to avoid confusion with the **-o** from System V (which has been  
26074 provided here as **-t** and as **-o** on XSI-conformant systems).

26075 The option list was significantly reduced from that provided by historical implementations.

26076 The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified  
26077 default output.

26078 It was recognized that mechanisms for dynamic linking make this utility less meaningful when  
26079 applied to an executable file (because a dynamically linked executable file may omit numerous  
26080 library routines that would be found in a statically linked executable file), but the value of *nm*  
26081 during software development was judged to outweigh other limitations.

26082 The default output format of *nm* is not specified because of differences in historical  
26083 implementations. The **-P** option was added to allow some type of portable output format. After  
26084 a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to  
26085 create one that did not match the current format of any of these four systems. The format  
26086 devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary  
26087 depending on locale (because no English descriptions are included). All of the systems currently  
26088 have the information available to use this format.

26089 The format given in *nm* STDOUT uses spaces between the fields, which may be any number of  
26090 <blank>s required to align the columns. The single-character types were selected to match  
26091 historical practice, and the requirement that implementation additions also be single characters  
26092 made parsing the information easier for shell scripts.

26093 **FUTURE DIRECTIONS**

26094 None.

26095 **SEE ALSO**26096 *ar*, *c99*26097 **CHANGE HISTORY**

26098 First released in Issue 2.

26099 **Issue 6**

26100 This utility is marked as supported when both the User Portability Utilities option and the  
26101 Software Development Utilities option are supported.

## 26102 NAME

26103           nohup — invoke a utility immune to hangups

## 26104 SYNOPSIS

26105           nohup *utility* [*argument...*]

## 26106 DESCRIPTION

26107           The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied  
26108           as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be  
26109           set to be ignored.

26110           If the standard output is a terminal, all output written by the named *utility* to its standard output  
26111           shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot  
26112           be created or opened for appending, the output shall be appended to the end of the file  
26113           **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be  
26114           created or opened for appending, *utility* shall not be invoked. If a file is created, the file's  
26115           permission bits shall be set to S\_IRUSR | S\_IWUSR.

26116           If the standard error is a terminal, all output written by the named *utility* to its standard error  
26117           shall be redirected to the same file descriptor as the standard output.

## 26118 OPTIONS

26119           None.

## 26120 OPERANDS

26121           The following operands shall be supported:

26122           *utility*           The name of a utility that is to be invoked. If the *utility* operand names any of the  
26123           special built-in utilities in Section 2.14 (on page 64), the results are undefined.

26124           *argument*       Any string to be supplied as an argument when invoking the utility named by the  
26125           *utility* operand.

## 26126 STDIN

26127           Not used.

## 26128 INPUT FILES

26129           None.

## 26130 ENVIRONMENT VARIABLES

26131           The following environment variables shall affect the execution of *nohup*:

26132           *HOME*           Determine the pathname of the user's home directory: if the output file **nohup.out**  
26133           cannot be created in the current directory, the *nohup* utility shall use the directory  
26134           named by *HOME* to create the file.

26135           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
26136           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
26137           Internationalization Variables for the precedence of internationalization variables  
26138           used to determine the values of locale categories.)

26139           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
26140           internationalization variables.

26141           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
26142           characters (for example, single-byte as opposed to multi-byte characters in  
26143           arguments).

26144           *LC\_MESSAGES*

26145           Determine the locale that should be used to affect the format and contents of



26146 diagnostic messages written to standard error.

26147 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

26148 **PATH** Determine the search path that is used to locate the utility to be invoked. See the  
 26149 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment  
 26150 Variables.

26151 **ASYNCHRONOUS EVENTS**

26152 The *nohup* utility shall take the standard action for all signals except that SIGHUP shall be  
 26153 ignored.

26154 **STDOUT**

26155 If the standard output is not a terminal, the standard output of *nohup* shall be the standard  
 26156 output generated by the execution of the *utility* specified by the operands. Otherwise, nothing  
 26157 shall be written to the standard output.

26158 **STDERR**

26159 If the standard output is a terminal, a message shall be written to the standard error, indicating  
 26160 the name of the file to which the output is being appended. The name of the file shall be either  
 26161 **nohup.out** or **\$HOME/nohup.out**.

26162 **OUTPUT FILES**

26163 If the standard output is a terminal, all output written by the named *utility* to the standard  
 26164 output and standard error is appended to the file **nohup.out**, which is created if it does not  
 26165 already exist.

26166 **EXTENDED DESCRIPTION**

26167 None.

26168 **EXIT STATUS**

26169 The following exit values shall be returned:

26170 126 The utility specified by *utility* was found but could not be invoked.

26171 127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be  
 26172 found.

26173 Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.

26174 **CONSEQUENCES OF ERRORS**

26175 Default.

26176 **APPLICATION USAGE**

26177 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
 26178 an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
 26179 utility exited with an error indication”. The value 127 was chosen because it is not commonly  
 26180 used for other meanings; most utilities use small values for “normal error conditions” and the  
 26181 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
 26182 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
 26183 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
 26184 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
 26185 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
 26186 any other reason.

26187 **EXAMPLES**

26188 It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by  
 26189 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and  
 26190 the *nohup* applies to everything in the file.

26191 Alternatively, the following command can be used to apply *nohup* to a complex command:

26192 `nohup sh -c 'complex-command-line'`

26193 **RATIONALE**

26194 The 4.3 BSD version ignores SIGTERM and SIGHUP, and if **./nohup.out** cannot be used, it fails  
26195 instead of trying to use **\$HOME/nohup.out**.

26196 The *cs*h utility has a built-in version of *nohup* that acts differently from the *nohup* defined in this  
26197 volume of IEEE Std 1003.1-2001.

26198 The term *utility* is used, rather than *command*, to highlight the fact that shell compound  
26199 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*  
26200 includes user application programs and shell scripts, not just the standard utilities.

26201 Historical versions of the *nohup* utility use default file creation semantics. Some more recent  
26202 versions use the permissions specified here as an added security precaution.

26203 Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore  
26204 SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several  
26205 reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this  
26206 volume of IEEE Std 1003.1-2001.

26207 **FUTURE DIRECTIONS**

26208 None.

26209 **SEE ALSO**

26210 Chapter 2 (on page 29), *sh*, the System Interfaces volume of IEEE Std 1003.1-2001, *signal()*

26211 **CHANGE HISTORY**

26212 First released in Issue 2.

26213 **NAME**

26214           od — dump files in various formats

26215 **SYNOPSIS**

26216           od [-v] [-A *address\_base*] [-j *skip*] [-N *count*] [-t *type\_string*] ...  
 26217           [*file*...]

26218 xSI       od [-bcdosx] [*file*] [[+] *offset* [.] [b]]

26219

26220 **DESCRIPTION**

26221       The *od* utility shall write the contents of its input files to standard output in a user-specified  
 26222       format.

26223 **OPTIONS**

26224       The *od* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 26225 xSI       Utility Syntax Guidelines, except that the order of presentation of the **-t** options and the  
 26226       **-bcdosx** options is significant.

26227       The following options shall be supported:

26228       **-A** *address\_base*

26229           Specify the input offset base. See the EXTENDED DESCRIPTION section. The  
 26230           application shall ensure that the *address\_base* option-argument is a character. The  
 26231           characters 'd', 'o', and 'x' specify that the offset base shall be written in  
 26232           decimal, octal, or hexadecimal, respectively. The character 'n' specifies that the  
 26233           offset shall not be written.

26234 xSI       **-b**       Interpret bytes in octal. This shall be equivalent to **-t o1**.

26235 xSI       **-c**       Interpret bytes as characters specified by the current setting of the *LC\_CTYPE*  
 26236           category. Certain non-graphic characters appear as C escapes: "NUL=\0",  
 26237           "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal  
 26238           numbers.

26239 xSI       **-d**       Interpret *words* (two-byte units) in unsigned decimal. This shall be equivalent to  
 26240           **-t u2**.

26241       **-j** *skip*       Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or  
 26242           seek past the first *skip* bytes in the concatenated input files. If the combined input  
 26243           is not at least *skip* bytes long, the *od* utility shall write a diagnostic message to  
 26244           standard error and exit with a non-zero exit status.

26245           By default, the *skip* option-argument shall be interpreted as a decimal number.  
 26246           With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number;  
 26247           otherwise, with a leading '0', the offset shall be interpreted as an octal number.  
 26248           Appending the character 'b', 'k', or 'm' to offset shall cause it to be interpreted  
 26249           as a multiple of 512, 1024, or 1048576 bytes, respectively. If the *skip* number is  
 26250           hexadecimal, any appended 'b' shall be considered to be the final hexadecimal  
 26251           digit.

26252       **-N** *count*      Format no more than *count* bytes of input. By default, *count* shall be interpreted as  
 26253           a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a  
 26254           hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an  
 26255           octal number. If *count* bytes of input (after successfully skipping, if **-j** *skip* is  
 26256           specified) are not available, it shall not be considered an error; the *od* utility shall  
 26257           format the input that is available.

|       |     |                        |                                                                                                                 |
|-------|-----|------------------------|-----------------------------------------------------------------------------------------------------------------|
| 26258 | XSI | <b>-o</b>              | Interpret <i>words</i> (two-byte units) in octal. This shall be equivalent to <b>-t o2</b> .                    |
| 26259 | XSI | <b>-s</b>              | Interpret <i>words</i> (two-byte units) in signed decimal. This shall be equivalent to                          |
| 26260 |     | <b>-t d2</b> .         |                                                                                                                 |
| 26261 |     | <b>-t type_string</b>  |                                                                                                                 |
| 26262 |     |                        | Specify one or more output types. See the EXTENDED DESCRIPTION section. The                                     |
| 26263 |     |                        | application shall ensure that the <i>type_string</i> option-argument is a string specifying                     |
| 26264 |     |                        | the types to be used when writing the input data. The string shall consist of the                               |
| 26265 |     |                        | type specification characters a, c, d, f, o, u, and x, specifying named character,                              |
| 26266 |     |                        | character, signed decimal, floating point, octal, unsigned decimal, and                                         |
| 26267 |     |                        | hexadecimal, respectively. The type specification characters d, f, o, u, and x can be                           |
| 26268 |     |                        | followed by an optional unsigned decimal integer that specifies the number of                                   |
| 26269 |     |                        | bytes to be transformed by each instance of the output type. The type specification                             |
| 26270 |     |                        | character f can be followed by an optional F, D, or L indicating that the conversion                            |
| 26271 |     |                        | should be applied to an item of type <b>float</b> , <b>double</b> , or <b>long double</b> , respectively.       |
| 26272 |     |                        | The type specification characters d, o, u, and x can be followed by an optional C, S,                           |
| 26273 |     |                        | I, or L indicating that the conversion should be applied to an item of type <b>char</b> ,                       |
| 26274 |     |                        | <b>short</b> , <b>int</b> , or <b>long</b> , respectively. Multiple types can be concatenated within the        |
| 26275 |     |                        | same <i>type_string</i> and multiple <b>-t</b> options can be specified. Output lines shall be                  |
| 26276 |     |                        | written for each type specified in the order in which the type specification                                    |
| 26277 |     |                        | characters are specified.                                                                                       |
| 26278 |     | <b>-v</b>              | Write all input data. Without the <b>-v</b> option, any number of groups of output lines,                       |
| 26279 |     |                        | which would be identical to the immediately preceding group of output lines                                     |
| 26280 |     |                        | (except for the byte offsets), shall be replaced with a line containing only an                                 |
| 26281 |     |                        | asterisk (' * ').                                                                                               |
| 26282 | XSI | <b>-x</b>              | Interpret <i>words</i> (two-byte units) in hexadecimal. This shall be equivalent to <b>-t x2</b> .              |
| 26283 | XSI |                        | Multiple types can be specified by using multiple <b>-bcdostx</b> options. Output lines are written for         |
| 26284 |     |                        | each type specified in the order in which the types are specified.                                              |
| 26285 |     | <b>OPERANDS</b>        |                                                                                                                 |
| 26286 |     |                        | The following operands shall be supported:                                                                      |
| 26287 |     | <i>file</i>            | A pathname of a file to be read. If no <i>file</i> operands are specified, the standard input                   |
| 26288 |     |                        | shall be used.                                                                                                  |
| 26289 |     |                        | If there are no more than two operands, none of the <b>-A</b> , <b>-j</b> , <b>-N</b> , or <b>-t</b> options is |
| 26290 |     |                        | specified, and either of the following is true: the first character of the last operand                         |
| 26291 |     |                        | is a plus sign (' + '), or there are two operands and the first character of the last                           |
| 26292 | XSI |                        | operand is numeric; the last operand shall be interpreted as an offset operand on                               |
| 26293 |     |                        | XSI-conformant systems. Under these conditions, the results are unspecified on                                  |
| 26294 |     |                        | systems that are not XSI-conformant systems.                                                                    |
| 26295 | XSI | <b>[+]offset[.][b]</b> | The <i>offset</i> operand specifies the offset in the file where dumping is to commence.                        |
| 26296 |     |                        | This operand is normally interpreted as octal bytes. If ' . ' is appended, the offset                           |
| 26297 |     |                        | shall be interpreted in decimal. If ' b ' is appended, the offset shall be interpreted                          |
| 26298 |     |                        | in units of 512 bytes.                                                                                          |
| 26299 |     | <b>STDIN</b>           |                                                                                                                 |
| 26300 |     |                        | The standard input shall be used only if no <i>file</i> operands are specified. See the INPUT FILES             |
| 26301 |     |                        | section.                                                                                                        |

26302 **INPUT FILES**

26303       The input files can be any file type.

26304 **ENVIRONMENT VARIABLES**

26305       The following environment variables shall affect the execution of *od*:

26306       **LANG**       Provide a default value for the internationalization variables that are unset or null.  
 26307                   (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 26308                   Internationalization Variables for the precedence of internationalization variables  
 26309                   used to determine the values of locale categories.)

26310       **LC\_ALL**     If set to a non-empty string value, override the values of all the other  
 26311                   internationalization variables.

26312       **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
 26313                   characters (for example, single-byte as opposed to multi-byte characters in  
 26314                   arguments and input files).

26315       **LC\_MESSAGES**  
 26316                   Determine the locale that should be used to affect the format and contents of  
 26317                   diagnostic messages written to standard error.

26318       **LC\_NUMERIC**  
 26319                   Determine the locale for selecting the radix character used when writing floating-  
 26320                   point formatted output.

26321 xSI     **NLSPATH**   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

26322 **ASYNCHRONOUS EVENTS**

26323       Default.

26324 **STDOUT**

26325       See the EXTENDED DESCRIPTION section.

26326 **STDERR**

26327       The standard error shall be used only for diagnostic messages.

26328 **OUTPUT FILES**

26329       None.

26330 **EXTENDED DESCRIPTION**

26331       The *od* utility shall copy sequentially each input file to standard output, transforming the input  
 26332 xSI       data according to the output types specified by the **-t** option or the **-bcdosx** options. If no  
 26333       output type is specified, the default output shall be as if **-t oS** had been specified.

26334       The number of bytes transformed by the output type specifier *c* may be variable depending on  
 26335       the *LC\_CTYPE* category.

26336       The default number of bytes transformed by output type specifiers *d*, *f*, *o*, *u*, and *x* corresponds  
 26337       to the various C-language types as follows. If the *c99* compiler is present on the system, these  
 26338       specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes  
 26339       may vary among systems that conform to IEEE Std 1003.1-2001.

- 26340       • For the type specifier characters *d*, *o*, *u*, and *x*, the default number of bytes shall correspond
- 26341       to the size of the underlying implementation's basic integer type. For these specifier
- 26342       characters, the implementation shall support values of the optional number of bytes to be
- 26343       converted corresponding to the number of bytes in the C-language types **char**, **short**, **int**, and
- 26344       **long**. These numbers can also be specified by an application as the characters '*C*', '*S*', '*I*',
- 26345       and '*L*', respectively. The implementation shall also support the values 1, 2, 4, and 8, even if
- 26346       it provides no C-Language types of those sizes. The implementation shall support the

decimal value corresponding to the C-language type **long long**. The byte order used when interpreting numeric values is implementation-defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

- For the type specifier character **f**, the default number of bytes shall correspond to the number of bytes in the underlying implementation's basic double precision floating-point data type. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types **float**, **double**, and **long double**. These numbers can also be specified by an application as the characters '**F**', '**D**', and '**L**', respectively.

The type specifier character **a** specifies that bytes shall be interpreted as named characters from the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least significant seven bits of each byte shall be used for this type specification. Bytes with the values listed in the following table shall be written using the corresponding names for those characters.

**Table 4-12** Named Characters in *od*

| Value | Name       | Value | Name       | Value | Name                     | Value | Name       |
|-------|------------|-------|------------|-------|--------------------------|-------|------------|
| \000  | <b>nul</b> | \001  | <b>soh</b> | \002  | <b>stx</b>               | \003  | <b>etx</b> |
| \004  | <b>eot</b> | \005  | <b>enq</b> | \006  | <b>ack</b>               | \007  | <b>bel</b> |
| \010  | <b>bs</b>  | \011  | <b>ht</b>  | \012  | <b>lf</b> or <b>nl</b> * | \013  | <b>vt</b>  |
| \014  | <b>ff</b>  | \015  | <b>cr</b>  | \016  | <b>so</b>                | \017  | <b>si</b>  |
| \020  | <b>dle</b> | \021  | <b>dc1</b> | \022  | <b>dc2</b>               | \023  | <b>dc3</b> |
| \024  | <b>dc4</b> | \025  | <b>nak</b> | \026  | <b>syn</b>               | \027  | <b>etb</b> |
| \030  | <b>can</b> | \031  | <b>em</b>  | \032  | <b>sub</b>               | \033  | <b>esc</b> |
| \034  | <b>fs</b>  | \035  | <b>gs</b>  | \036  | <b>rs</b>                | \037  | <b>us</b>  |
| \040  | <b>sp</b>  | \177  | <b>del</b> |       |                          |       |            |

**Note:** The "\012" value may be written either as **lf** or **nl**.

The type specifier character **c** specifies that bytes shall be interpreted as characters specified by the current setting of the *LC\_CTYPE* locale category. Characters listed in the table in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be written as the corresponding escape sequences, except that backslash shall be written as a single backslash and a NUL shall be written as '\0'. Other non-printable characters shall be written as one three-digit octal number for each byte in the character. If the size of a byte on the system is greater than nine bits, the format used for non-printable characters is implementation-defined. Printable multi-byte characters shall be written in the area corresponding to the first byte of the character; the two-character sequence "\*\*\*" shall be written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. When either the **-j skip** or **-N count** option is specified along with the **c** type specifier, and this results in an attempt to start or finish in the middle of a multi-byte character, the result is implementation-defined.

The input data shall be manipulated in blocks, where a block is defined as a multiple of the least common multiple of the number of bytes transformed by the specified output types. If the least common multiple is greater than 16, the results are unspecified. Each input block shall be written as transformed by each output type, one per written line, in the order that the output types were specified. If the input block size is larger than the number of bytes transformed by the output type, the output type shall sequentially transform the parts of the input block, and the output from each of the transformations shall be separated by one or more <blank>s.

26393 If, as a result of the specification of the **-N** option or end-of-file being reached on the last input  
 26394 file, input data only partially satisfies an output type, the input shall be extended sufficiently  
 26395 with null bytes to write the last byte of the input.

26396 Unless **-A n** is specified, the first output line produced for each input block shall be preceded by  
 26397 the input offset, cumulative across input files, of the next byte to be written. The format of the  
 26398 input offset is unspecified; however, it shall not contain any <blank>s, shall start at the first  
 26399 character of the output line, and shall be followed by one or more <blank>s. In addition, the  
 26400 offset of the byte following the last byte written shall be written after all the input data has been  
 26401 processed, but shall not be followed by any <blank>s.

26402 If no **-A** option is specified, the input offset base is unspecified.

## 26403 EXIT STATUS

26404 The following exit values shall be returned:

26405 0 All input files were processed successfully.

26406 >0 An error occurred.

## 26407 CONSEQUENCES OF ERRORS

26408 Default.

## 26409 APPLICATION USAGE

26410 XSI-conformant applications are warned not to use filenames starting with '+' or a first  
 26411 operand starting with a numeric character so that the old functionality can be maintained by  
 26412 implementations, unless they specify one of the **-A**, **-j**, or **-N** options. To guarantee that one of  
 26413 these filenames is always interpreted as a filename, an application could always specify the  
 26414 address base format with the **-A** option.

## 26415 EXAMPLES

26416 If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as  
 26417 standard input to the command:

26418 `od -A d -t a`

26419 on an implementation using an input block size of 16 bytes, the standard output, independent of  
 26420 the current locale setting, would be similar to:

```
26421 00000000 nul soh stx etx eot enq ack bel bs ht nl vt ff cr so si
26422 0000016 dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc fs gs rs us
26423 0000032 sp ! " # $ % & ' () * + , - . /
26424 0000048 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
26425 0000064 @ A B C D E F G H I J K L M N O
26426 0000080 P Q R S T U V W X Y Z [\] ^ _
26427 0000096 ` a b c d e f g h i j k l m n o
26428 0000112 p q r s t u v w x y z { | } ~ del
26429 0000128
```

26430 Note that this volume of IEEE Std 1003.1-2001 allows **nl** or **lf** to be used as the name for the  
 26431 ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character  
 26432 **lf** (line feed), but traditional implementations have referred to this character as newline (**nl**) and  
 26433 the POSIX locale character set symbolic name for the corresponding character is a <newline>.

26434 The command:

26435 `od -A o -t o2x2x -N 18`

26436 on a system with 32-bit words and an implementation using an input block size of 16 bytes  
 26437 could write 18 bytes in approximately the following format:

```

26438 00000000 032056 031440 041123 042040 052516 044530 020043 031464
26439 342e 3320 4253 4420 554e 4958 2023 3334
26440 342e3320 42534420 554e4958 20233334
26441 00000020 032472
26442 353a
26443 353a0000
26444 00000022

```

26445 The command:

```
26446 od -A d -t f -t o4 -t x4 -N 24 -j 0x15
```

26447 on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point  
 26448 format) would skip 21 bytes of input data and then write 24 bytes in approximately the  
 26449 following format:

```

26450 00000000 1.0000000000000000e+00 1.5735000000000000e+01
26451 07774000000 000000000000 10013674121 35341217270
26452 3ff00000 00000000 402f3851 eb851eb8
26453 00000016 1.4066823000000000e+02
26454 10030312542 04370303230
26455 40619562 23e18698
26456 00000024

```

#### 26457 RATIONALE

26458 The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently  
 26459 *hexdump*. There were several objections to all of these based on the following reasons:

- 26460 • The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- 26461 • The *hexdump* description was much more complex than needed for a simple dump utility.
- 26462 • The *od* utility has been available on all historical implementations and there was no need to  
 26463 create a new name for a utility so similar to the historical *od* utility.

26464 The original reasons for not standardizing historical *od* were also fairly widespread. Those  
 26465 reasons are given below along with rationale explaining why the standard developers believe  
 26466 that this version does not suffer from the indicated problem:

- 26467 • The BSD and System V versions of *od* have diverged, and the intersection of features  
 26468 provided by both does not meet the needs of the user community. In fact, the System V  
 26469 version only provides a mechanism for dumping octal bytes and **shorts**, signed and unsigned  
 26470 decimal **shorts**, hexadecimal **shorts**, and ASCII characters. BSD added the ability to dump  
 26471 **floats**, **doubles**, named ASCII characters, and octal, signed decimal, unsigned decimal, and  
 26472 hexadecimal **longs**. The version presented here provides more normalized forms for  
 26473 dumping bytes, **shorts**, **ints**, and **longs** in octal, signed decimal, unsigned decimal, and  
 26474 hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as current locale  
 26475 characters.
- 26476 • It would not be possible to come up with a compatible superset of the BSD and System V  
 26477 flags that met the requirements of the standard developers. The historical default *od* output is  
 26478 the specified default output of this utility. None of the option letters chosen for this version  
 26479 of *od* conflict with any of the options to historical versions of *od*.
- 26480 • On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps  
 26481 of **ints**, even in the BSD version. Because of the way options are named, the name space  
 26482 could not be extended to solve these problems. This is why the **-t** option was added (with  
 26483 type specifiers more closely matched to the *printf()* formats used in the rest of this volume of



IEEE Std 1003.1-2001) and the optional field sizes were added to the `d`, `f`, `o`, `u`, and `x` type specifiers. It is also one of the reasons why the historical practice was not mandated as a required obsolescent form of `od`. (Although the old versions of `od` are not listed as an obsolescent form, implementations are urged to continue to recognize the older forms for several more years.) The `a`, `c`, `f`, `o`, and `x` types match the meaning of the corresponding format characters in the historical implementations of `od` except for the default sizes of the fields converted. The `d` format is signed in this volume of IEEE Std 1003.1-2001 to match the `printf()` notation. (Historical versions of `od` used `d` as a synonym for `u` in this version. The System V implementation uses `s` for signed decimal; BSD uses `i` for signed decimal and `s` for null-terminated strings.) Other than `d` and `u`, all of the type specifiers match format characters in the historical BSD version of `od`.

The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are used even though it is recognized that there may be zero or more than one compiler for the C language on an implementation and that they may use different sizes for some of these types. (For example, one compiler might use 2 bytes **shorts**, 2 bytes **ints**, and 4 bytes **longs**, while another compiler (or an option to the same compiler) uses 2 bytes **shorts**, 4 bytes **ints**, and 4 bytes **longs**.) Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values reported by invocations of the `getconf` utility when called with `system_var` operands {`UCHAR_MAX`}, {`USHORT_MAX`}, {`UINT_MAX`}, and {`ULONG_MAX`} for the types **char**, **short**, **int**, and **long**, respectively. There are similar constants required by the ISO C standard, but not required by the System Interfaces volume of IEEE Std 1003.1-2001 or this volume of IEEE Std 1003.1-2001. They are {`FLT_MANT_DIG`}, {`DBL_MANT_DIG`}, and {`LDBL_MANT_DIG`} for the types **float**, **double**, and **long double**, respectively. If the optional `c99` utility is provided by the implementation and used as specified by this volume of IEEE Std 1003.1-2001, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guarantee that the `od` utility is able to interpret binary data output by such a program correctly.

This volume of IEEE Std 1003.1-2001 requires that the numeric values of these lengths be recognized by the `od` utility and that symbolic forms also be recognized. Thus, a conforming application can always look at an array of **unsigned long** data elements using `od -t uL`.

- The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The `-A` option now specifies the address base and the `-S` option specifies a starting offset.
- It would be difficult to break the dependence on U.S. ASCII to achieve an internationalized utility. It does not seem to be any harder for `od` to dump characters in the current locale than it is for the `ed` or `sed` **I** commands. The `c` type specifier does this without difficulty and is completely compatible with the historical implementations of the `c` format character when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The `a` type specifier (from the BSD `a` format character) was left as a portable means to dump ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by `pax` could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard as a subset of their base codeset.

The use of `"**"` as an indication of continuation of a multi-byte character in `c` specifier output was chosen based on seeing an implementation that uses this method. The continuation bytes have to be marked in a way that is not ambiguous with another single-byte or multi-byte character.

An early proposal used `-S` and `-n`, respectively, for the `-j` and `-N` options eventually selected. These were changed to avoid conflicts with historical implementations.

26532 The original standard specified **-t o2** as the default when no output type was given. This was  
 26533 changed to **-t oS** (the length of a **short**) to accommodate a supercomputer implementation that  
 26534 historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not  
 26535 affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at  
 26536 the same time to address an historical implementation that had no two-byte data types in its C  
 26537 compiler.

26538 The use of a basic integer data type is intended to allow the implementation to choose a word  
 26539 size commonly used by applications on that architecture.

#### 26540 **FUTURE DIRECTIONS**

26541 All option and operand interfaces marked as extensions may be withdrawn in a future version.

#### 26542 **SEE ALSO**

26543 *c99, sed*

#### 26544 **CHANGE HISTORY**

26545 First released in Issue 2.

#### 26546 **Issue 5**

26547 In the description of the **-c** option, the phrase “This is equivalent to **-t c.**” is deleted.

26548 The FUTURE DIRECTIONS section is modified.

#### 26549 **Issue 6**

26550 The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the  
 26551 revisions in the IEEE P1003.2b draft standard.

26552 The normative text is reworded to avoid use of the term “must” for application requirements.

26553 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/33 is applied, correcting the examples |  
 26554 which used an undefined **-n** option, which should have been **-N**. |

26555 **NAME**

26556           paste — merge corresponding or subsequent lines of files

26557 **SYNOPSIS**26558           paste [-s] [-d *list*] *file...*26559 **DESCRIPTION**26560           The *paste* utility shall concatenate the corresponding lines of the given input files, and write the  
26561           resulting lines to standard output.26562           The default operation of *paste* shall concatenate the corresponding lines of the input files. The  
26563           <newline> of every line except the line from the last input file shall be replaced with a <tab>.26564           If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall  
26565           behave as though empty lines were read from the files on which end-of-file was detected, unless  
26566           the **-s** option is specified.26567 **OPTIONS**26568           The *paste* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
26569           12.2, Utility Syntax Guidelines.

26570           The following options shall be supported:

26571           **-d list**       Unless a backslash character appears in *list*, each character in *list* is an element  
26572                           specifying a delimiter character. If a backslash character appears in *list*, the  
26573                           backslash character and one or more characters following it are an element  
26574                           specifying a delimiter character as described below. These elements specify one or  
26575                           more delimiters to use, instead of the default <tab>, to replace the <newline> of  
26576                           the input lines. The elements in *list* shall be used circularly; that is, when the list is  
26577                           exhausted the first element from the list is reused. When the **-s** option is specified:

- 26578
- The last <newline> in a file shall not be modified.
  - The delimiter shall be reset to the first element of *list* after each *file* operand is  
26579                           processed.
  - 26580

26581           When the **-s** option is not specified:

- 26582
- The <newline>s in the file specified by the last *file* operand shall not be  
26583                           modified.
  - The delimiter shall be reset to the first element of *list* each time a line is  
26584                           processed from each file.
  - 26585

26586           If a backslash character appears in *list*, it and the character following it shall be  
26587           used to represent the following delimiter characters:26588           \ *n*   <newline>.26589           \ *t*   <tab>.

26590           \\   Backslash character.

26591           \0   Empty string (not a null character). If '*\0*' is immediately followed by the  
26592                   character '*x*', the character '*X*', or any character defined by the *LC\_CTYPE*  
26593                   **digit** keyword (see the Base Definitions volume of IEEE Std 1003.1-2001,  
26594                   Chapter 7, Locale), the results are unspecified.

26595           If any other characters follow the backslash, the results are unspecified.

26596           **-s**           Concatenate all of the lines of each separate input file in command line order. The  
26597                   <newline> of every line except the last line in each input file shall be replaced with

26598 the <tab>, unless otherwise specified by the **-d** option.

## 26599 OPERANDS

26600 The following operand shall be supported:

26601 *file* A pathname of an input file. If '-' is specified for one or more of the *files*, the  
 26602 standard input shall be used; the standard input shall be read one line at a time,  
 26603 circularly, for each instance of '-'. Implementations shall support pasting of at  
 26604 least 12 *file* operands.

## 26605 STDIN

26606 The standard input shall be used only if one or more *file* operands is '-'. See the INPUT FILES  
 26607 section.

## 26608 INPUT FILES

26609 The input files shall be text files, except that line lengths shall be unlimited.

## 26610 ENVIRONMENT VARIABLES

26611 The following environment variables shall affect the execution of *paste*:

26612 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 26613 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 26614 Internationalization Variables for the precedence of internationalization variables  
 26615 used to determine the values of locale categories.)

26616 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 26617 internationalization variables.

26618 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 26619 characters (for example, single-byte as opposed to multi-byte characters in  
 26620 arguments and input files).

26621 *LC\_MESSAGES*  
 26622 Determine the locale that should be used to affect the format and contents of  
 26623 diagnostic messages written to standard error.

26624 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 26625 ASYNCHRONOUS EVENTS

26626 Default.

## 26627 STDOUT

26628 Concatenated lines of input files shall be separated by the <tab> (or other characters under the  
 26629 control of the **-d** option) and terminated by a <newline>.

## 26630 STDERR

26631 The standard error shall be used only for diagnostic messages.

## 26632 OUTPUT FILES

26633 None.

## 26634 EXTENDED DESCRIPTION

26635 None.

## 26636 EXIT STATUS

26637 The following exit values shall be returned:

26638 0 Successful completion.

26639 >0 An error occurred.

26640 **CONSEQUENCES OF ERRORS**

26641 If one or more input files cannot be opened when the **-s** option is not specified, a diagnostic  
 26642 message shall be written to standard error, but no output is written to standard output. If the **-s**  
 26643 option is specified, the *paste* utility shall provide the default behavior described in Section 1.11  
 26644 (on page 20).

26645 **APPLICATION USAGE**

26646 When the escape sequences of the *list* option-argument are used in a shell script, they must be  
 26647 quoted; otherwise, the shell treats the `'\'` as a special character.

26648 Conforming applications should only use the specific backslash escaped delimiters presented in  
 26649 this volume of IEEE Std 1003.1-2001. Historical implementations treat `'\x'`, where `'x'` is not in  
 26650 this list, as `'x'`, but future implementations are free to expand this list to recognize other  
 26651 common escapes similar to those accepted by *printf* and other standard utilities.

26652 Most of the standard utilities work on text files. The *cut* utility can be used to turn files with  
 26653 arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used  
 26654 to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
26655 cut -b 1-500 -n file > file1
26656 cut -b 501- -n file > file2
```

26657 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that  
 26658 contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in  
 26659 *file* that are longer than 500 + {LINE\_MAX} bytes. The original file can be recreated from **file1**  
 26660 and **file2** using the command:

```
26661 paste -d "\0" file1 file2 > file
```

26662 The commands:

```
26663 paste -d "\0" ...
26664 paste -d "" ...
```

26665 are not necessarily equivalent; the latter is not specified by this volume of IEEE Std 1003.1-2001  
 26666 and may result in an error. The construct `'\0'` is used to mean “no separator” because  
 26667 historical versions of *paste* did not follow the syntax guidelines, and the command:

```
26668 paste -d "" ...
```

26669 could not be handled properly by *getopt()*.

26670 **EXAMPLES**

26671 1. Write out a directory in four columns:

```
26672 ls | paste - - - -
```

26673 2. Combine pairs of lines from a file into single lines:

```
26674 paste -s -d "\t\n" file
```

26675 **RATIONALE**

26676 None.

26677 **FUTURE DIRECTIONS**

26678 None.

26679 **SEE ALSO**26680           Section 1.11 (on page 20), *cut*, *grep*, *pr*26681 **CHANGE HISTORY**

26682           First released in Issue 2.

26683 **Issue 6**

26684           The normative text is reworded to avoid use of the term “must” for application requirements.

26685 **NAME**

26686 patch — apply changes to files

26687 **SYNOPSIS**

```
26688 UP patch [-blNR] [-c | -e | -n] [-d dir] [-D define] [-i patchfile]
26689 [-o outfile] [-p num] [-r rejectfile] [file]
```

26690

26691 **DESCRIPTION**

26692 The *patch* utility shall read a source (patch) file containing any of the three forms of difference  
 26693 (diff) listings produced by the *diff* utility (normal, context, or in the style of *ed*) and apply those  
 26694 differences to a file. By default, *patch* shall read from the standard input.

26695 The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a **-c**,  
 26696 **-e**, or **-n** option.

26697 If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they  
 26698 came from separate patch files. (In this case, the application shall ensure that the name of the  
 26699 patch file is determinable for each *diff* listing.)

26700 **OPTIONS**

26701 The *patch* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 26702 12.2, Utility Syntax Guidelines.

26703 The following options shall be supported:

26704 **-b** Save a copy of the original contents of each modified file, before the differences are  
 26705 applied, in a file of the same name with the suffix **.orig** appended to it. If the file  
 26706 already exists, it shall be overwritten; if multiple patches are applied to the same  
 26707 file, the **.orig** file shall be written only for the first patch. When the **-o outfile** option  
 26708 is also specified, *file.orig* shall not be created but, if *outfile* already exists,  
 26709 *outfile.orig* shall be created.

26710 **-c** Interpret the patch file as a context difference (the output of the utility *diff* when  
 26711 the **-c** or **-C** options are specified).

26712 **-d dir** Change the current directory to *dir* before processing as described in the  
 26713 EXTENDED DESCRIPTION section.

26714 **-D define** Mark changes with one of the following C preprocessor constructs:

```
26715 #ifdef define
26716 ...
26717 #endif
```

```
26718 #ifndef define
26719 ...
26720 #endif
```

26721 optionally combined with the C preprocessor construct **#else**. If the patched file is  
 26722 processed with the C preprocessor, where the macro *define* is defined, the output  
 26723 shall contain the changes from the patch file; otherwise, the output shall not  
 26724 contain the patches specified in the patch file.

26725 **-e** Interpret the patch file as an *ed* script, rather than a *diff* script.

26726 **-i patchfile** Read the patch information from the file named by the pathname *patchfile*, rather  
 26727 than the standard input.

|       |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 26728 | <b>-l</b>                   | (The letter ell.) Cause any sequence of <blank>s in the difference script to match any sequence of <blank>s in the input file. Other characters shall be matched exactly.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 26731 | <b>-n</b>                   | Interpret the script as a normal difference.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 26732 | <b>-N</b>                   | Ignore patches where the differences have already been applied to the file; by default, already-applied patches shall be rejected.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 26734 | <b>-o <i>outfile</i></b>    | Instead of modifying the files (specified by the <i>file</i> operand or the difference listings) directly, write a copy of the file referenced by each patch, with the appropriate differences applied, to <i>outfile</i> . Multiple patches for a single file shall be applied to the intermediate versions of the file created by any previous patches, and shall result in multiple, concatenated versions of the file being written to <i>outfile</i> .                                                                                                                                                                                                                    |
| 26740 | <b>-p <i>num</i></b>        | For all pathnames in the patch file that indicate the names of files to be patched, delete <i>num</i> pathname components from the beginning of each pathname. If the pathname in the patch file is absolute, any leading slashes shall be considered the first component (that is, <b>-p 1</b> shall remove the leading slashes). Specifying <b>-p 0</b> shall cause the full pathname to be used. If <b>-p</b> is not specified, only the basename (the final pathname component) shall be used.                                                                                                                                                                             |
| 26746 | <b>-R</b>                   | Reverse the sense of the patch script; that is, assume that the difference script was created from the new version to the old version. The <b>-R</b> option cannot be used with <i>ed</i> scripts. The <i>patch</i> utility shall attempt to reverse each portion of the script before applying it. Rejected differences shall be saved in swapped format. If this option is not specified, and until a portion of the patch file is successfully applied, <i>patch</i> attempts to apply each portion in its reversed sense as well as in its normal sense. If the attempt is successful, the user shall be prompted to determine whether the <b>-R</b> option should be set. |
| 26754 | <b>-r <i>rejectfile</i></b> | Override the default reject filename. In the default case, the reject file shall have the same name as the output file, with the suffix <b>.rej</b> appended to it; see <b>Patch Application</b> (on page 694).                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

## 26757 OPERANDS

26758 The following operand shall be supported:

26759 *file* A pathname of a file to patch.

## 26760 STDIN

26761 See the INPUT FILES section.

## 26762 INPUT FILES

26763 Input files shall be text files.

## 26764 ENVIRONMENT VARIABLES

26765 The following environment variables shall affect the execution of *patch*:

26766 *LANG* Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

26770 *LC\_ALL* If set to a non-empty string value, override the values of all the other internationalization variables.



26772        **LC\_CTYPE**    Determine the locale for the interpretation of sequences of bytes of text data as  
 26773                   characters (for example, single-byte as opposed to multi-byte characters in  
 26774                   arguments and input files).

26775        **LC\_MESSAGES**  
 26776                   Determine the locale that should be used to affect the format and contents of  
 26777                   diagnostic messages written to standard error and informative messages written to  
 26778                   standard output.

26779 xSI        **NLSPATH**    Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

26780        **LC\_TIME**    Determine the locale for recognizing the format of file timestamps written by the  
 26781                   *diff* utility in a context-difference input file.

26782 **ASYNCHRONOUS EVENTS**  
 26783        Default.

26784 **STDOUT**  
 26785        Not used.

26786 **STDERR**  
 26787        The standard error shall be used for diagnostic and informational messages.

26788 **OUTPUT FILES**  
 26789        The output of the *patch* utility, the save files (**.orig** suffixes), and the reject files (**.rej** suffixes)  
 26790        shall be text files.

26791 **EXTENDED DESCRIPTION**  
 26792        A patch file may contain patching instructions for more than one file; filenames shall be  
 26793        determined as specified in **Filename Determination** (on page 694). When the **-b** option is  
 26794        specified, for each patched file, the original shall be saved in a file of the same name with the  
 26795        suffix **.orig** appended to it.

26796        For each patched file, a reject file may also be created as noted in **Patch Application** (on page  
 26797        694). In the absence of a **-r** option, the name of this file shall be formed by appending the suffix  
 26798        **.rej** to the original filename.

26799        **Patch File Format**

26800        The patch file shall contain zero or more lines of header information followed by one or more  
 26801        patches. Each patch shall contain zero or more lines of filename identification in the format  
 26802        produced by *diff -c*, and one or more sets of *diff* output, which are customarily called *hunks*.

26803        The *patch* utility shall recognize the following expression in the header information:

26804        **Index:** *pathname*  
 26805                   The file to be patched is named *pathname*.

26806        If all lines (including headers) within a patch begin with the same leading sequence of <blank>s,  
 26807        the *patch* utility shall remove this sequence before proceeding. Within each patch, if the type of  
 26808        difference is context, the *patch* utility shall recognize the following expressions:

26809        \*\*\* *filename timestamp*  
 26810                   The patches arose from *filename*.

26811        --- *filename timestamp*  
 26812                   The patches should be applied to *filename*.

26813        Each hunk within a patch shall be the *diff* output to change a line range within the original file.  
 26814        The line numbers for successive hunks within a patch shall occur in ascending order.

**Filename Determination**

If no *file* operand is specified, *patch* shall perform the following steps to determine the filename to use:

1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as specified by the **-p** option) from the filename on the line beginning with "\*\*\*\*", then test for the existence of this file relative to the current directory (or the directory specified with the **-d** option). If the file exists, the *patch* utility shall use this filename.
2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as specified by the **-p** option) from the filename on the line beginning with "---", then test for the existence of this file relative to the current directory (or the directory specified with the **-d** option). If the file exists, the *patch* utility shall use this filename.
3. If the header information contains a line beginning with the string **Index:**, the *patch* utility shall delete pathname components (as specified by the **-p** option) from this line, then test for the existence of this file relative to the current directory (or the directory specified with the **-d** option). If the file exists, the *patch* utility shall use this filename.
4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a **get -e SCCS/s.filename** command to retrieve an editable version of the file. If the file exists, the *patch* utility shall use this filename.
5. The *patch* utility shall write a prompt to standard output and request a filename interactively from the controlling terminal (for example, **/dev/tty**).

**Patch Application**

If the **-c**, **-e**, or **-n** option is present, the *patch* utility shall interpret information within each hunk as a context difference, an *ed* difference, or a normal difference, respectively. In the absence of any of these options, the *patch* utility shall determine the type of difference based on the format of information within the hunk.

For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line number at the beginning of the hunk, plus or minus any offset used in applying the previous hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and backwards at least 1 000 bytes for a set of lines that match the hunk context.

If no such place is found and it is a context difference, then another scan shall take place, ignoring the first and last line of context. If that fails, the first two and last two lines of context shall be ignored and another scan shall be made. Implementations may search more extensively for installation locations.

If no location can be found, the *patch* utility shall append the hunk to the reject file. The rejected hunk shall be written in context-difference format regardless of the format of the patch file. If the input was a normal or *ed*-style difference, the reject file may contain differences with zero lines of context. The line numbers on the hunks in the reject file may be different from the line numbers in the patch file since they shall reflect the approximate locations for the failed hunks in the new file rather than the old one.

If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking the *ed* utility.

**EXIT STATUS**

The following exit values shall be returned:

- 0 Successful completion.

26859           1   One or more lines were written to a reject file.

26860           >1   An error occurred.

## 26861 CONSEQUENCES OF ERRORS

26862           Patches that cannot be correctly placed in the file shall be written to a reject file.

## 26863 APPLICATION USAGE

26864           The **-R** option does not work with *ed* scripts because there is too little information to reconstruct the reverse operation.

26866           The **-p** option makes it possible to customize a patch file to local user directory structures without manually editing the patch file. For example, if the filename in the patch file was:

26868           /*curds/whey/src/blurfl/blurfl.c*

26869           Setting **-p 0** gives the entire pathname unmodified; **-p 1** gives:

26870           *curds/whey/src/blurfl/blurfl.c*

26871           without the leading slash, **-p 4** gives:

26872           *blurfl/blurfl.c*

26873           and not specifying **-p** at all gives:

26874           *blurfl.c* .

## 26875 EXAMPLES

26876           None.

## 26877 RATIONALE

26878           Some of the functionality in historical *patch* implementations was not specified. The following documents those features present in historical implementations that have not been specified.

26880           A deleted piece of functionality was the '+' pseudo-option allowing an additional set of options and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

26882           In historical implementations, if the string "Prereq:" appeared in the header, the *patch* utility would search for the corresponding version information (the string specified in the header, delimited by <blank>s or the beginning or end of a line or the file) anywhere in the original file. This was deleted as too simplistic and insufficiently trustworthy a mechanism to standardize. For example, if:

26887           Prereq: 1.2

26888           were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the prerequisite.

26890           The following options were dropped from historical implementations of *patch* as insufficiently useful to standardize:

26892           **-b**           The **-b** option historically provided a method for changing the name extension of the backup file from the default **.orig**. This option has been modified and retained in this volume of IEEE Std 1003.1-2001.

26895           **-F**           The **-F** option specified the number of lines of a context diff to ignore when searching for a place to install a patch.

26897           **-f**           The **-f** option historically caused *patch* not to request additional information from the user.

|       |                          |                                                                                                                                   |
|-------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| 26899 | <b>-r</b>                | The <b>-r</b> option historically provided a method of overriding the extension of the reject file from the default <b>.rej</b> . |
| 26900 |                          |                                                                                                                                   |
| 26901 | <b>-s</b>                | The <b>-s</b> option historically caused <i>patch</i> to work silently unless an error occurred.                                  |
| 26902 | <b>-x</b>                | The <b>-x</b> option historically set internal debugging flags.                                                                   |
| 26903 |                          | In some file system implementations, the saving of a <b>.orig</b> file may produce unwanted results. In                           |
| 26904 |                          | the case of 12, 13, or 14-character filenames (on file systems supporting 14-character maximum                                    |
| 26905 |                          | filenames), the <b>.orig</b> file overwrites the new file. The reject file may also exceed this filename                          |
| 26906 |                          | limit. It was suggested, due to some historical practice, that a tilde ('~') suffix be used instead                               |
| 26907 |                          | of <b>.orig</b> and some other character instead of the <b>.rej</b> suffix. This was rejected because it is not                   |
| 26908 |                          | obvious to the user which file is which. The suffixes <b>.orig</b> and <b>.rej</b> are clearer and more                           |
| 26909 |                          | understandable.                                                                                                                   |
| 26910 |                          | The <b>-b</b> option has the opposite sense in some historical implementations—do not save the <b>.orig</b>                       |
| 26911 |                          | file. The default case here is not to save the files, making <i>patch</i> behave more consistently with the                       |
| 26912 |                          | other standard utilities.                                                                                                         |
| 26913 |                          | The <b>-w</b> option in early proposals was changed to <b>-I</b> to match historical practice.                                    |
| 26914 |                          | The <b>-N</b> option was included because without it, a non-interactive application cannot reject                                 |
| 26915 |                          | previously applied patches. For example, if a user is piping the output of <i>diff</i> into the <i>patch</i>                      |
| 26916 |                          | utility, and the user only wants to patch a file to a newer version non-interactively, the <b>-N</b>                              |
| 26917 |                          | option is required.                                                                                                               |
| 26918 |                          | Changes to the <b>-I</b> option description were proposed to allow matching across <newline>s in                                  |
| 26919 |                          | addition to just <blank>s. Since this is not historical practice, and since some ambiguities could                                |
| 26920 |                          | result, it is suggested that future developments in this area utilize another option letter, such as                              |
| 26921 |                          | <b>-L</b> .                                                                                                                       |
| 26922 | <b>FUTURE DIRECTIONS</b> |                                                                                                                                   |
| 26923 |                          | None.                                                                                                                             |
| 26924 | <b>SEE ALSO</b>          |                                                                                                                                   |
| 26925 |                          | <i>ed</i> , <i>diff</i>                                                                                                           |
| 26926 | <b>CHANGE HISTORY</b>    |                                                                                                                                   |
| 26927 |                          | First released in Issue 4.                                                                                                        |
| 26928 | <b>Issue 5</b>           |                                                                                                                                   |
| 26929 |                          | The FUTURE DIRECTIONS section is added.                                                                                           |
| 26930 | <b>Issue 6</b>           |                                                                                                                                   |
| 26931 |                          | This utility is marked as part of the User Portability Utilities option.                                                          |
| 26932 |                          | The description of the <b>-D</b> option and the steps in <b>Filename Determination</b> (on page 694) are                          |
| 26933 |                          | changed to match historical practice as defined in the IEEE P1003.2b draft standard.                                              |
| 26934 |                          | The normative text is reworded to avoid use of the term “must” for application requirements.                                      |
| 26935 |                          | IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/34 is applied, clarifying the way that the                                       |
| 26936 |                          | <i>patch</i> utility performs <b>ifdef</b> selection for the <b>-D</b> option.                                                    |

26937 **NAME**

26938 pathchk — check pathnames

26939 **SYNOPSIS**26940 pathchk [-p] *pathname...*26941 **DESCRIPTION**

26942 The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used  
 26943 to access or create a file without causing syntax errors) and portable (that is, no filename  
 26944 truncation results). More extensive portability checks are provided by the **-p** option.

26945 By default, the *pathchk* utility shall check each component of each *pathname* operand based on the  
 26946 underlying file system. A diagnostic shall be written for each *pathname* operand that:

- 26947 • Is longer than {PATH\_MAX} bytes (see **Pathname Variable Values** in the Base Definitions  
 26948 volume of IEEE Std 1003.1-2001, Chapter 13, Headers, <limits.h>)
- 26949 • Contains any component longer than {NAME\_MAX} bytes in its containing directory
- 26950 • Contains any component in a directory that is not searchable
- 26951 • Contains any character in any component that is not valid in its containing directory

26952 The format of the diagnostic message is not specified, but shall indicate the error detected and  
 26953 the corresponding *pathname* operand.

26954 It shall not be considered an error if one or more components of a *pathname* operand do not exist  
 26955 as long as a file matching the pathname specified by the missing components could be created  
 26956 that does not violate any of the checks specified above.

26957 **OPTIONS**

26958 The *pathchk* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 26959 12.2, Utility Syntax Guidelines.

26960 The following option shall be supported:

- 26961 **-p** Instead of performing checks based on the underlying file system, write a  
 26962 diagnostic for each *pathname* operand that:
  - 26963 • Is longer than {\_POSIX\_PATH\_MAX} bytes (see **Minimum Values** in the Base  
 26964 Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers, <limits.h>)
  - 26965 • Contains any component longer than {\_POSIX\_NAME\_MAX} bytes
  - 26966 • Contains any character in any component that is not in the portable filename  
 26967 character set

26968 **OPERANDS**

26969 The following operand shall be supported:

26970 *pathname* A pathname to be checked.

26971 **STDIN**

26972 Not used.

26973 **INPUT FILES**

26974 None.

26975 **ENVIRONMENT VARIABLES**

26976 The following environment variables shall affect the execution of *pathchk*:

26977 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 26978 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,

26979 Internationalization Variables for the precedence of internationalization variables  
 26980 used to determine the values of locale categories.)

26981 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 26982 internationalization variables.

26983 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 26984 characters (for example, single-byte as opposed to multi-byte characters in  
 26985 arguments).

26986 *LC\_MESSAGES*  
 26987 Determine the locale that should be used to affect the format and contents of  
 26988 diagnostic messages written to standard error.

26989 *XS1* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

26990 **ASYNCHRONOUS EVENTS**  
 26991 Default.

26992 **STDOUT**  
 26993 Not used.

26994 **STDERR**  
 26995 The standard error shall be used only for diagnostic messages.

26996 **OUTPUT FILES**  
 26997 None.

26998 **EXTENDED DESCRIPTION**  
 26999 None.

27000 **EXIT STATUS**  
 27001 The following exit values shall be returned:  
 27002 0 All *pathname* operands passed all of the checks.  
 27003 >0 An error occurred.

27004 **CONSEQUENCES OF ERRORS**  
 27005 Default.

27006 **APPLICATION USAGE**  
 27007 The *test* utility can be used to determine whether a given *pathname* names an existing file; it  
 27008 does not, however, give any indication of whether or not any component of the *pathname* was  
 27009 truncated in a directory where the *\_POSIX\_NO\_TRUNC* feature is not in effect. The *pathchk*  
 27010 utility does not check for file existence; it performs checks to determine whether a *pathname*  
 27011 does exist or could be created with no *pathname* component truncation.

27012 The *noclobber* option in the shell (see the *set* special built-in) can be used to atomically create a  
 27013 file. As with all file creation semantics in the System Interfaces volume of IEEE Std 1003.1-2001,  
 27014 it guarantees atomic creation, but still depends on applications to agree on conventions and  
 27015 cooperate on the use of files after they have been created.

27016 **EXAMPLES**  
 27017 To verify that all *pathnames* in an imported data interchange archive are legitimate and  
 27018 unambiguous on the current system:  
 27019 

```
pax -f archive | sed -e '/' == './s///' | xargs pathchk
```

  
 27020 

```
if [$? -eq 0]
```

  
 27021 

```
then
```

  
 27022 

```
 pax -r -f archive
```

```

27023 else
27024 echo Investigate problems before importing files.
27025 exit 1
27026 fi

27027 To verify that all files in the current directory hierarchy could be moved to any system
27028 conforming to the System Interfaces volume of IEEE Std 1003.1-2001 that also supports the pax
27029 utility:

27030 find . -print | xargs pathchk -p
27031 if [$? -eq 0]
27032 then
27033 pax -w -f archive .
27034 else
27035 echo Portable archive cannot be created.
27036 exit 1
27037 fi

27038 To verify that a user-supplied pathname names a readable file and that the application can create
27039 a file extending the given path without truncation and without overwriting any existing file:

27040 case $- in
27041 *C*) reset="";;
27042 *) reset="set +C"
27043 set -C;;
27044 esac
27045 test -r "$path" && pathchk "$path.out" &&
27046 rm "$path.out" > "$path.out"
27047 if [$? -ne 0]; then
27048 printf "%s: %s not found or %s.out fails \
27049 creation checks.\n" $0 "$path" "$path"
27050 $reset # Reset the noclobber option in case a trap
27051 # on EXIT depends on it.
27052 exit 1
27053 fi
27054 $reset
27055 PROCESSING < "$path" > "$path.out"

27056 The following assumptions are made in this example:

27057 1. PROCESSING represents the code that is used by the application to use $path once it is
27058 verified that $path.out works as intended.

27059 2. The state of the noclobber option is unknown when this code is invoked and should be set
27060 on exit to the state it was in when this code was invoked. (The reset variable is used in this
27061 example to restore the initial state.)

27062 3. Note the usage of:

27063 rm "$path.out" > "$path.out"

27064 a. The pathchk command has already verified, at this point, that $path.out is not
27065 truncated.

27066 b. With the noclobber option set, the shell verifies that $path.out does not already exist
27067 before invoking rm.

```

27068 c. If the shell succeeded in creating **\$path.out**, *rm* removes it so that the application can  
 27069 create the file again in the **PROCESSING** step.

27070 d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:

27071 `rm "$path.out" > "$path.out"`

27072 should be replaced with:

27073 `> "$path.out"`

27074 which verifies that the file did not already exist, but leaves **\$path.out** in place for use  
 27075 by **PROCESSING**.

#### 27076 RATIONALE

27077 The *pathchk* utility was new for the ISO POSIX-2:1993 standard. It, along with the *set*  
 27078 **-C(noclobber)** option added to the shell, replaces the *mktemp*, *validfnam*, and *create* utilities that  
 27079 appeared in early proposals. All of these utilities were attempts to solve several common  
 27080 problems:

- 27081 • Verify the validity (for several different definitions of “valid”) of a pathname supplied by a  
 27082 user, generated by an application, or imported from an external source.
- 27083 • Atomically create a file.
- 27084 • Perform various string handling functions to generate a temporary filename.

27085 The *create* utility, included in an early proposal, provided checking and atomic creation in a  
 27086 single invocation of the utility; these are orthogonal issues and need not be grouped into a single  
 27087 utility. Note that the *noclobber* option also provides a way of creating a lock for process  
 27088 synchronization; since it provides an atomic *create*, there is no race between a test for existence  
 27089 and the following creation if it did not exist.

27090 Having a function like *tmpnam()* in the ISO C standard is important in many high-level  
 27091 languages. The shell programming language, however, has built-in string manipulation  
 27092 facilities, making it very easy to construct temporary filenames. The names needed obviously  
 27093 depend on the application, but are frequently of a form similar to:

27094 `$TMPDIR/application_abbreviation$.suffix`

27095 In cases where there is likely to be contention for a given suffix, a simple shell **for** or **while** loop  
 27096 can be used with the shell *noclobber* option to create a file without risk of collisions, as long as  
 27097 applications trying to use the same filename name space are cooperating on the use of files after  
 27098 they have been created.

#### 27099 FUTURE DIRECTIONS

27100 None.

#### 27101 SEE ALSO

27102 Section 2.7 (on page 43), *set* (on page 86), *test*

#### 27103 CHANGE HISTORY

27104 First released in Issue 4.



27105 **NAME**

27106       pax — portable archive interchange

27107 **SYNOPSIS**27108       pax [-cdnv] [-H|-L] [-f *archive*] [-s *replstr*]... [*pattern*...]27109       pax -r[-cdiknuv] [-H|-L] [-f *archive*] [-o *options*]... [-p *string*]...  
27110       [-s *replstr*]... [*pattern*...]27111       pax -w[-dituvX] [-H|-L] [-b *blocksize*] [[-a] [-f *archive*] [-o *options*]...  
27112       [-s *replstr*]... [-x *format*] [*file*...]27113       pax -r -w[-diklntuvX] [-H|-L] [-p *string*]... [-s *replstr*]...  
27114       [*file*...] *directory*27115 **DESCRIPTION**27116       The *pax* utility shall read, write, and write lists of the members of archive files and copy  
27117       directory hierarchies. A variety of archive formats shall be supported; see the **-x** *format* option.27118       The action to be taken depends on the presence of the **-r** and **-w** options. The four combinations  
27119       of **-r** and **-w** are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes,  
27120       corresponding respectively to the four forms shown in the SYNOPSIS section.27121       **list**       In **list** mode (when neither **-r** nor **-w** are specified), *pax* shall write the names of  
27122       the members of the archive file read from the standard input, with pathnames  
27123       matching the specified patterns, to standard output. If a named file is of type  
27124       directory, the file hierarchy rooted at that file shall be listed as well.27125       **read**       In **read** mode (when **-r** is specified, but **-w** is not), *pax* shall extract the members of  
27126       the archive file read from the standard input, with pathnames matching the  
27127       specified patterns. If an extracted file is of type directory, the file hierarchy rooted  
27128       at that file shall be extracted as well. The extracted files shall be created performing  
27129       pathname resolution with the directory in which *pax* was invoked as the current  
27130       working directory.27131       If an attempt is made to extract a directory when the directory already exists, this  
27132       shall not be considered an error. If an attempt is made to extract a FIFO when the  
27133       FIFO already exists, this shall not be considered an error.27134       The ownership, access, and modification times, and file mode of the restored files  
27135       are discussed under the **-p** option.27136       **write**       In **write** mode (when **-w** is specified, but **-r** is not), *pax* shall write the contents of  
27137       the *file* operands to the standard output in an archive format. If no *file* operands are  
27138       specified, a list of files to copy, one per line, shall be read from the standard input.  
27139       A file of type directory shall include all of the files in the file hierarchy rooted at the  
27140       file.27141       **copy**       In **copy** mode (when both **-r** and **-w** are specified), *pax* shall copy the *file* operands  
27142       to the destination directory.27143       If no *file* operands are specified, a list of files to copy, one per line, shall be read  
27144       from the standard input. A file of type directory shall include all of the files in the  
27145       file hierarchy rooted at the file.27146       The effect of the **copy** shall be as if the copied files were written to an archive file  
27147       and then subsequently extracted, except that there may be hard links between the  
27148       original and the copied files. If the destination directory is a subdirectory of one of  
27149       the files to be copied, the results are unspecified. If the destination directory is a

27150 file of a type not defined by the System Interfaces volume of IEEE Std 1003.1-2001,  
 27151 the results are implementation-defined; otherwise, it shall be an error for the file  
 27152 named by the *directory* operand not to exist, not be writable by the user, or not be a  
 27153 file of type directory.

27154 In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member,  
 27155 *pax* shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces  
 27156 volume of IEEE Std 1003.1-2001, called with the following arguments:

- 27157 • The intermediate directory used as the *path* argument
- 27158 • The value of the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO as the *mode*  
 27159 argument

27160 If any specified *pattern* or *file* operands are not matched by at least one file or archive member,  
 27161 *pax* shall write a diagnostic message to standard error for each one that did not match and exit  
 27162 with a non-zero exit status.

27163 The archive formats described in the EXTENDED DESCRIPTION section shall be automatically  
 27164 detected on input. The default output archive format shall be implementation-defined.

27165 A single archive can span multiple files. The *pax* utility shall determine, in an implementation-  
 27166 defined manner, what file to read or write as the next file.

27167 If the selected archive format supports the specification of linked files, it shall be an error if these  
 27168 files cannot be linked when the archive is extracted. For archive formats that do not store file  
 27169 contents with each name that causes a hard link, if the file that contains the data is not extracted  
 27170 during this *pax* session, either the data shall be restored from the original file, or a diagnostic  
 27171 message shall be displayed with the name of a file that can be used to extract the data. In  
 27172 traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited  
 27173 directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall  
 27174 write a diagnostic message to standard error and shall terminate.

## 27175 OPTIONS

27176 The *pax* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 27177 12.2, Utility Syntax Guidelines, except that the order of presentation of the **-o**, **-p**, and **-s** options  
 27178 is significant.

27179 The following options shall be supported:

- 27180 **-r** Read an archive file from standard input.
- 27181 **-w** Write files to the standard output in the specified archive format.
- 27182 **-a** Append files to the end of the archive. It is implementation-defined which devices  
 27183 on the system support appending. Additional file formats unspecified by this  
 27184 volume of IEEE Std 1003.1-2001 may impose restrictions on appending.
- 27185 **-b** *blocksize* Block the output at a positive decimal integer number of bytes per write to the  
 27186 archive file. Devices and archive formats may impose restrictions on blocking.  
 27187 Blocking shall be automatically determined on input. Conforming applications  
 27188 shall not specify a *blocksize* value larger than 32 256. Default blocking when  
 27189 creating archives depends on the archive format. (See the **-x** option below.)
- 27190 **-c** Match all file or archive members except those specified by the *pattern* or *file*  
 27191 operands.
- 27192 **-d** Cause files of type directory being copied or archived or archive members of type  
 27193 directory being extracted or listed to match only the file or archive member itself  
 27194 and not the file hierarchy rooted at the file.

|       |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27195 | <b>-f</b> <i>archive</i> | Specify the pathname of the input or output archive, overriding the default standard input (in <b>list</b> or <b>read</b> modes) or standard output ( <b>write</b> mode).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 27196 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27197 | <b>-H</b>                | If a symbolic link referencing a file of type directory is specified on the command line, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line, then <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior shall be to archive the symbolic link itself.                                                                                                                                                                                                                                                                           |
| 27198 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27199 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27200 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27201 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27202 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27203 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27204 | <b>-i</b>                | Interactively rename files or archive members. For each archive member matching a <i>pattern</i> operand or file matching a <i>file</i> operand, a prompt shall be written to the file <b>/dev/tty</b> . The prompt shall contain the name of the file or archive member, but the format is otherwise unspecified. A line shall then be read from <b>/dev/tty</b> . If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The <i>pax</i> utility shall immediately exit with a non-zero exit status if end-of-file is encountered when reading a response or if <b>/dev/tty</b> cannot be opened for reading and writing. |
| 27205 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27206 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27207 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27208 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27209 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27210 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27211 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27212 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27213 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27214 |                          | The results of extracting a hard link to a file that has been renamed during extraction are unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27215 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27216 | <b>-k</b>                | Prevent the overwriting of existing files.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27217 | <b>-l</b>                | (The letter ell.) In <b>copy</b> mode, hard links shall be made between the source and destination file hierarchies whenever possible. If specified in conjunction with <b>-H</b> or <b>-L</b> , when a symbolic link is encountered, the hard link created in the destination file hierarchy shall be to the file referenced by the symbolic link. If specified when neither <b>-H</b> nor <b>-L</b> is specified, when a symbolic link is encountered, the implementation shall create a hard link to the symbolic link in the source file hierarchy or copy the symbolic link to the destination.                                                                                                                                                                                                                                   |
| 27218 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27219 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27220 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27221 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27222 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27223 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27224 | <b>-L</b>                | If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior shall be to archive the symbolic link itself.                                                                                                                                                                |
| 27225 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27226 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27227 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27228 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27229 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27230 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27231 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27232 | <b>-n</b>                | Select the first archive member that matches each <i>pattern</i> operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27233 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27234 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27235 | <b>-o</b> <i>options</i> | Provide information to the implementation to modify the algorithm for extracting or writing files. The value of <i>options</i> shall consist of one or more comma-separated keywords of the form:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27236 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27237 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27238 |                          | <i>keyword</i> [[:]= <i>value</i> ] [, <i>keyword</i> [[:]= <i>value</i> ] , ...]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27239 |                          | Some keywords apply only to certain file formats, as indicated with each description. Use of keywords that are inapplicable to the file format being processed produces undefined results.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27240 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27241 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

Keywords in the *options* argument shall be a string that would be a valid portable filename as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.276, Portable Filename Character Set.

**Note:** Keywords are not expected to be filenames, merely to follow the same character composition rules as portable filenames.

Keywords can be preceded with white space. The *value* field shall consist of zero or more characters; within *value*, the application shall precede any literal comma with a backslash, which shall be ignored, but preserves the comma as part of *value*. A comma as the final character, or a comma followed solely by white space as the final characters, in *options* shall be ignored. Multiple **-o** options can be specified; if keywords given to these multiple **-o** options conflict, the keywords and values appearing later in command line sequence shall take precedence and the earlier shall be silently ignored. The following keyword values of *options* shall be supported for the file formats as indicated:

**delete=pattern**

(Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax* shall omit from extended header records that it produces any keywords matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore any keywords matching the string pattern in the extended header records. In both cases, matching shall be performed using the pattern matching notation described in Section 2.13.1 (on page 62) and Section 2.13.2 (on page 63). For example:

**-o delete=security.\***

would suppress security-related information. See **pax Extended Header** (on page 714) for extended header record keyword usage.

**exthdr.name=string**

(Applicable only to the **-x pax** format.) This keyword allows user control over the name that is written into the **ustar** header blocks for the extended header produced under the circumstances described in **pax Header Block** (on page 713). The name shall be the contents of *string*, after the following character substitutions have been made:

| <i>string</i><br><b>Includes:</b> | <b>Replaced By:</b>                                                                                                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------|
| %d                                | The directory name of the file, equivalent to the result of the <i>dirname</i> utility on the translated pathname. |
| %f                                | The filename of the file, equivalent to the result of the <i>basename</i> utility on the translated pathname.      |
| %p                                | The process ID of the <i>pax</i> process.                                                                          |
| %%                                | A '%' character.                                                                                                   |

Any other '%' characters in *string* produce undefined results.

If no **-o exthdr.name=string** is specified, *pax* shall use the following default value:

%d/PaxHeaders.%p/%f

**globexthdr.name=string**

(Applicable only to the **-x pax** format.) When used in **write** or **copy** mode with the appropriate options, *pax* shall create global extended header records

with **ustar** header blocks that will be treated as regular files by previous versions of *pax*. This keyword allows user control over the name that is written into the **ustar** header blocks for global extended header records. The name shall be the contents of string, after the following character substitutions have been made:

| <i>string</i><br><b>Includes:</b> | <b>Replaced By:</b>                                                                                                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------|
| %n                                | An integer that represents the sequence number of the global extended header record in the archive, starting at 1. |
| %p                                | The process ID of the <i>pax</i> process.                                                                          |
| %%                                | A ' % ' character.                                                                                                 |

Any other ' % ' characters in *string* produce undefined results.

If no **-o globexthdr.name=string** is specified, *pax* shall use the following default value:

\$TMPDIR/GlobalHead.%p.%n

where *\$TMPDIR* represents the value of the *TMPDIR* environment variable. If *TMPDIR* is not set, *pax* shall use **/tmp**.

#### **invalid=action**

(Applicable only to the **-x pax** format.) This keyword allows user control over the action *pax* takes upon encountering values in an extended header record that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list** mode, cannot be written in the codeset and current locale of the implementation. The following are invalid values that shall be recognized by *pax*:

- In **read** or **copy** mode, a filename or link name that contains character encodings invalid in the destination hierarchy. (For example, the name may contain embedded NULs.)
- In **read** or **copy** mode, a filename or link name that is longer than the maximum allowed in the destination hierarchy (for either a pathname component or the entire pathname).
- In **list** mode, any character string value (filename, link name, user name, and so on) that cannot be written in the codeset and current locale of the implementation.

The following mutually-exclusive values of the *action* argument are supported:

- bypass** In **read** or **copy** mode, *pax* shall bypass the file, causing no change to the destination hierarchy. In **list** mode, *pax* shall write all requested valid values for the file, but its method for writing invalid values is unspecified.
- rename** In **read** or **copy** mode, *pax* shall act as if the **-i** option were in effect for each file with invalid filename or link name values, allowing the user to provide a replacement name interactively. In **list** mode, *pax* shall behave identically to the **bypass** action.
- UTF-8** When used in **read**, **copy**, or **list** mode and a filename, link name, owner name, or any other field in an extended header

27333 record cannot be translated from the **pax** UTF-8 codeset format  
 27334 to the codeset and current locale of the implementation, *pax*  
 27335 shall use the actual UTF-8 encoding for the name.

27336 **write** In **read** or **copy** mode, *pax* shall write the file, translating or  
 27337 truncating the name, regardless of whether this may overwrite  
 27338 an existing file with a valid name. In **list** mode, *pax* shall behave  
 27339 identically to the **bypass** action.

27340 If no **-o invalid=** option is specified, *pax* shall act as if **-o invalid=bypass** were  
 27341 specified. Any overwriting of existing files that may be allowed by the  
 27342 **-o invalid=** actions shall be subject to permission (**-p**) and modification time  
 27343 (**-u**) restrictions, and shall be suppressed if the **-k** option is also specified.

27344 **linkdata**  
 27345 (Applicable only to the **-x pax** format.) In **write** mode, *pax* shall write the  
 27346 contents of a file to the archive even when that file is merely a hard link to a  
 27347 file whose contents have already been written to the archive.

27348 **listopt=format**  
 27349 This keyword specifies the output format of the table of contents produced  
 27350 when the **-v** option is specified in **list** mode. See **List Mode Format**  
 27351 **Specifications** (on page 709). To avoid ambiguity, the **listopt=format** shall be  
 27352 the only or final **keyword=value** pair in a **-o** option-argument; all characters in  
 27353 the remainder of the option-argument shall be considered part of the format  
 27354 string. When multiple **-olistopt=format** options are specified, the format  
 27355 strings shall be considered a single, concatenated string, evaluated in  
 27356 command line order.

27357 **times**  
 27358 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax* |  
 27359 shall include **atime**, **ctime**, and **mtime** extended header records for each file.  
 27360 See **pax Extended Header File Times** (on page 717).

27361 In addition to these keywords, if the **-x pax** format is specified, any of the |  
 27362 keywords and values defined in **pax Extended Header** (on page 714), including  
 27363 implementation extensions, can be used in **-o** option-arguments, in either of two  
 27364 modes:

27365 **keyword=value**  
 27366 When used in **write** or **copy** mode, these keyword/value pairs shall be  
 27367 included at the beginning of the archive as **typeflag g** global extended header  
 27368 records. When used in **read** or **list** mode, these keyword/value pairs shall act  
 27369 as if they had been at the beginning of the archive as **typeflag g** global  
 27370 extended header records.

27371 **keyword:=value**  
 27372 When used in **write** or **copy** mode, these keyword/value pairs shall be  
 27373 included as records at the beginning of a **typeflag x** extended header for each  
 27374 file. (This shall be equivalent to the equal-sign form except that it creates no  
 27375 **typeflag g** global extended header records.) When used in **read** or **list** mode,  
 27376 these keyword/value pairs shall act as if they were included as records at the  
 27377 end of each extended header; thus, they shall override any global or file-  
 27378 specific extended header record keywords of the same names. For example, in  
 27379 the command:

```

27380 pax -r -o "
27381 gname:=mygroup,
27382 " <archive

```

27383 the group name will be forced to a new value for all files read from the  
 27384 archive.

27385 The precedence of **-o** keywords over various fields in the archive is described in  
 27386 **pax Extended Header Keyword Precedence** (on page 717).

27387 **-p *string*** Specify one or more file characteristic options (privileges). The *string* option-  
 27388 argument shall be a string specifying file characteristics to be retained or discarded  
 27389 on extraction. The string shall consist of the specification characters a, e, m, o, and  
 27390 p. Other implementation-defined characters can be included. Multiple  
 27391 characteristics can be concatenated within the same string and multiple **-p** options  
 27392 can be specified. The meaning of the specification characters are as follows:

- 27393 a Do not preserve file access times.
- 27394 e Preserve the user ID, group ID, file mode bits (see the Base Definitions volume  
 27395 of IEEE Std 1003.1-2001, Section 3.168, File Mode Bits), access time,  
 27396 modification time, and any other implementation-defined file characteristics.
- 27397 m Do not preserve file modification times.
- 27398 o Preserve the user ID and group ID.
- 27399 p Preserve the file mode bits. Other implementation-defined file mode attributes  
 27400 may be preserved.

27401 In the preceding list, “preserve” indicates that an attribute stored in the archive  
 27402 shall be given to the extracted file, subject to the permissions of the invoking  
 27403 process. The access and modification times of the file shall be preserved unless  
 27404 otherwise specified with the **-p** option or not stored in the archive. All attributes  
 27405 that are not preserved shall be determined as part of the normal file creation action  
 27406 (see Section 1.7.1.4 (on page 4)).

27407 If neither the e nor the o specification character is specified, or the user ID and  
 27408 group ID are not preserved for any reason, *pax* shall not set the S\_ISUID and  
 27409 S\_ISGID bits of the file mode.

27410 If the preservation of any of these items fails for any reason, *pax* shall write a  
 27411 diagnostic message to standard error. Failure to preserve these items shall affect  
 27412 the final exit status, but shall not cause the extracted file to be deleted.

27413 If file characteristic letters in any of the *string* option-arguments are duplicated or  
 27414 conflict with each other, the ones given last shall take precedence. For example, if  
 27415 **-p eme** is specified, file modification times are preserved.

27416 **-s *replstr*** Modify file or archive member names named by *pattern* or *file* operands according  
 27417 to the substitution expression *replstr*, using the syntax of the *ed* utility. The  
 27418 concepts of “address” and “line” are meaningless in the context of the *pax* utility,  
 27419 and shall not be supplied. The format shall be:

```

27420 -s /old/new/ [gp]

```

27421 where as in *ed*, *old* is a basic regular expression and *new* can contain an ampersand,  
 27422 ‘\n’ (where *n* is a digit) backreferences, or subexpression matching. The *old* string  
 27423 shall also be permitted to contain <newline>s.

|       |                  |                                                                                                           |
|-------|------------------|-----------------------------------------------------------------------------------------------------------|
| 27424 |                  | Any non-null character can be used as a delimiter ('/' shown here). Multiple <b>-s</b>                    |
| 27425 |                  | expressions can be specified; the expressions shall be applied in the order                               |
| 27426 |                  | specified, terminating with the first successful substitution. The optional trailing                      |
| 27427 |                  | 'g' is as defined in the <i>ed</i> utility. The optional trailing 'p' shall cause successful              |
| 27428 |                  | substitutions to be written to standard error. File or archive member names that                          |
| 27429 |                  | substitute to the empty string shall be ignored when reading and writing archives.                        |
| 27430 | <b>-t</b>        | When reading files from the file system, and if the user has the permissions                              |
| 27431 |                  | required by <i>utime()</i> to do so, set the access time of each file read to the access time             |
| 27432 |                  | that it had before being read by <i>pax</i> .                                                             |
| 27433 | <b>-u</b>        | Ignore files that are older (having a less recent file modification time) than a pre-                     |
| 27434 |                  | existing file or archive member with the same name. In <b>read</b> mode, an archive                       |
| 27435 |                  | member with the same name as a file in the file system shall be extracted if the                          |
| 27436 |                  | archive member is newer than the file. In <b>write</b> mode, an archive file member with                  |
| 27437 |                  | the same name as a file in the file system shall be superseded if the file is newer                       |
| 27438 |                  | than the archive member. If <b>-a</b> is also specified, this is accomplished by appending                |
| 27439 |                  | to the archive; otherwise, it is unspecified whether this is accomplished by actual                       |
| 27440 |                  | replacement in the archive or by appending to the archive. In <b>copy</b> mode, the file in               |
| 27441 |                  | the destination hierarchy shall be replaced by the file in the source hierarchy or by                     |
| 27442 |                  | a link to the file in the source hierarchy if the file in the source hierarchy is newer.                  |
| 27443 | <b>-v</b>        | In <b>list</b> mode, produce a verbose table of contents (see the STDOUT section).                        |
| 27444 |                  | Otherwise, write archive member pathnames to standard error (see the STDERR                               |
| 27445 |                  | section).                                                                                                 |
| 27446 | <b>-x format</b> | Specify the output archive format. The <i>pax</i> utility shall support the following                     |
| 27447 |                  | formats:                                                                                                  |
| 27448 | <b>cpio</b>      | The <b>cpio</b> interchange format; see the EXTENDED DESCRIPTION                                          |
| 27449 |                  | section. The default <i>blocksize</i> for this format for character special                               |
| 27450 |                  | archive files shall be 5120. Implementations shall support all                                            |
| 27451 |                  | <i>blocksize</i> values less than or equal to 32 256 that are multiples of 512.                           |
| 27452 | <b>pax</b>       | The <b>pax</b> interchange format; see the EXTENDED DESCRIPTION                                           |
| 27453 |                  | section. The default <i>blocksize</i> for this format for character special                               |
| 27454 |                  | archive files shall be 5120. Implementations shall support all                                            |
| 27455 |                  | <i>blocksize</i> values less than or equal to 32 256 that are multiples of 512.                           |
| 27456 | <b>ustar</b>     | The <b>tar</b> interchange format; see the EXTENDED DESCRIPTION                                           |
| 27457 |                  | section. The default <i>blocksize</i> for this format for character special                               |
| 27458 |                  | archive files shall be 10 240. Implementations shall support all                                          |
| 27459 |                  | <i>blocksize</i> values less than or equal to 32 256 that are multiples of 512.                           |
| 27460 |                  | Implementation-defined formats shall specify a default block size as well as any                          |
| 27461 |                  | other block sizes supported for character special archive files.                                          |
| 27462 |                  | Any attempt to append to an archive file in a format different from the existing                          |
| 27463 |                  | archive format shall cause <i>pax</i> to exit immediately with a non-zero exit status.                    |
| 27464 |                  | In <b>copy</b> mode, if no <b>-x</b> format is specified, <i>pax</i> shall behave as if <b>-xpax</b> were |
| 27465 |                  | specified.                                                                                                |
| 27466 | <b>-X</b>        | When traversing the file hierarchy specified by a pathname, <i>pax</i> shall not descend                  |
| 27467 |                  | into directories that have a different device ID ( <i>st_dev</i> ; see the System Interfaces              |
| 27468 |                  | volume of IEEE Std 1003.1-2001, <i>stat()</i> ).                                                          |



27469 The options that operate on the names of files or archive members (**-c**, **-i**, **-n**, **-s**, **-u**, and **-v**)  
 27470 shall interact as follows. In **read** mode, the archive members shall be selected based on the user-  
 27471 specified *pattern* operands as modified by the **-c**, **-n**, and **-u** options. Then, any **-s** and **-i** options  
 27472 shall modify, in that order, the names of the selected files. The **-v** option shall write names  
 27473 resulting from these modifications.

27474 In **write** mode, the files shall be selected based on the user-specified pathnames as modified by  
 27475 the **-n** and **-u** options. Then, any **-s** and **-i** options shall modify, in that order, the names of  
 27476 these selected files. The **-v** option shall write names resulting from these modifications.

27477 If both the **-u** and **-n** options are specified, *pax* shall not consider a file selected unless it is newer  
 27478 than the file to which it is compared.

## 27479 List Mode Format Specifications

27480 In **list** mode with the **-o listopt=format** option, the *format* argument shall be applied for each  
 27481 selected file. The *pax* utility shall append a <newline> to the **listopt** output for each selected file.  
 27482 The *format* argument shall be used as the *format* string described in the Base Definitions volume  
 27483 of IEEE Std 1003.1-2001, Chapter 5, File Format Notation, with the exceptions 1. through 5.  
 27484 defined in the EXTENDED DESCRIPTION section of *printf*, plus the following exceptions:

27485 6. The sequence (*keyword*) can occur before a format conversion specifier. The conversion  
 27486 argument is defined by the value of *keyword*. The implementation shall support the  
 27487 following keywords:

27488 — Any of the Field Name entries in Table 4-13 (on page 718) and Table 4-15 (on page 721).  
 27489 The implementation may support the *cpio* keywords without the leading **c\_** in addition  
 27490 to the form required by Table 4-16 (on page 722).

27491 — Any keyword defined for the extended header in **pax Extended Header** (on page 714).

27492 — Any keyword provided as an implementation-defined extension within the extended  
 27493 header defined in **pax Extended Header** (on page 714).

27494 For example, the sequence "%(charset)s" is the string value of the name of the character  
 27495 set in the extended header.

27496 The result of the keyword conversion argument shall be the value from the applicable  
 27497 header field or extended header, without any trailing NULs.

27498 All keyword values used as conversion arguments shall be translated from the UTF-8  
 27499 encoding to the character set appropriate for the local file system, user database, and so on,  
 27500 as applicable.

27501 7. An additional conversion specifier character, T, shall be used to specify time formats. The T  
 27502 conversion specifier character can be preceded by the sequence (*keyword=subformat*), where  
 27503 *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime**  
 27504 and the default subformat shall be:

27505 %b %e %H:%M %Y

27506 8. An additional conversion specifier character, M, shall be used to specify the file mode string  
 27507 as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used.  
 27508 For example, % . 1M writes the single character corresponding to the <entry type> field of the  
 27509 *ls -l* command.

27510 9. An additional conversion specifier character, D, shall be used to specify the device for block  
 27511 or special files, if applicable, in an implementation-defined format. If not applicable, and  
 27512 (*keyword*) is specified, then this conversion shall be equivalent to % (*keyword*) u. If not

- 27513 applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to <space>.
- 27514 10. An additional conversion specifier character, F, shall be used to specify a pathname. The F  
27515 conversion character can be preceded by a sequence of comma-separated keywords:
- 27516 (*keyword* [, *keyword*] . . . )
- 27517 The values for all the keywords that are non-null shall be concatenated together, each  
27518 separated by a '/'. The default shall be (**path**) if the keyword **path** is defined; otherwise,  
27519 the default shall be (**prefix,name**).
- 27520 11. An additional conversion specifier character, L, shall be used to specify a symbolic line  
27521 expansion. If the current file is a symbolic link, then %L shall expand to:
- 27522 "%s -> %s", <value of keyword>, <contents of link>
- 27523 Otherwise, the %L conversion specification shall be the equivalent of %F.

## 27524 OPERANDS

27525 The following operands shall be supported:

- 27526 *directory* The destination directory pathname for **copy** mode.
- 27527 *file* A pathname of a file to be copied or archived.
- 27528 *pattern* A pattern matching one or more pathnames of archive members. A pattern must  
27529 be given in the name-generating notation of the pattern matching notation in  
27530 Section 2.13 (on page 62), including the filename expansion rules in Section 2.13.3  
27531 (on page 63). The default, if no *pattern* is specified, is to select all members in the  
27532 archive.

## 27533 STDIN

- 27534 In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a  
27535 text file containing a list of pathnames, one per line, without leading or trailing <blank>s.
- 27536 In **list** and **read** modes, if **-f** is not specified, the standard input shall be an archive file.
- 27537 Otherwise, the standard input shall not be used.

## 27538 INPUT FILES

- 27539 The input file named by the *archive* option-argument, or standard input when the archive is read  
27540 from there, shall be a file formatted according to one of the specifications in the EXTENDED  
27541 DESCRIPTION section or some other implementation-defined format.
- 27542 The file **/dev/tty** shall be used to write prompts and read responses.

## 27543 ENVIRONMENT VARIABLES

- 27544 The following environment variables shall affect the execution of *pax*:
- 27545 *LANG* Provide a default value for the internationalization variables that are unset or null.  
27546 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
27547 Internationalization Variables for the precedence of internationalization variables  
27548 used to determine the values of locale categories.)
- 27549 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
27550 internationalization variables.
- 27551 *LC\_COLLATE*  
27552 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
27553 character collating elements used in the pattern matching expressions for the  
27554 *pattern* operand, the basic regular expression for the **-s** option, and the extended  
27555 regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES*

27556 category.

27557 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 27558 characters (for example, single-byte as opposed to multi-byte characters in  
 27559 arguments and input files), the behavior of character classes used in the extended  
 27560 regular expression defined for the **yesexpr** locale keyword in the **LC\_MESSAGES**  
 27561 category, and pattern matching.

27562 **LC\_MESSAGES**  
 27563 Determine the locale for the processing of affirmative responses that should be  
 27564 used to affect the format and contents of diagnostic messages written to standard  
 27565 error.

27566 **LC\_TIME** Determine the format and contents of date and time strings when the **-v** option is  
 27567 specified.

27568 xsi **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

27569 **TMPDIR** Determine the pathname that provides part of the default global extended header  
 27570 record file, as described for the **-o globexthdr=** keyword in the **OPTIONS** section.

27571 **TZ** Determine the timezone used to calculate date and time strings when the **-v** option  
 27572 is specified. If **TZ** is unset or null, an unspecified default timezone shall be used.

27573 **ASYNCHRONOUS EVENTS**

27574 Default.

27575 **STDOUT**

27576 In **write** mode, if **-f** is not specified, the standard output shall be the archive formatted  
 27577 according to one of the specifications in the **EXTENDED DESCRIPTION** section, or some other  
 27578 implementation-defined format (see **-x format**).

27579 In **list** mode, when the **-olistopt=format** has been specified, the selected archive members shall  
 27580 be written to standard output using the format described under **List Mode Format**  
 27581 **Specifications** (on page 709). In **list** mode without the **-olistopt=format** option, the table of  
 27582 contents of the selected archive members shall be written to standard output using the following  
 27583 format:

27584 "%s\n", <pathname>

27585 If the **-v** option is specified in **list** mode, the table of contents of the selected archive members  
 27586 shall be written to standard output using the following formats.

27587 For pathnames representing hard links to previous members of the archive:

27588 "%sΔ=Δ%s\n", <ls -l listing>, <linkname>

27589 For all other pathnames:

27590 "%s\n", <ls -l listing>

27591 where <ls -l listing> shall be the format specified by the **ls** utility with the **-l** option. When  
 27592 writing pathnames in this format, it is unspecified what is written for fields for which the  
 27593 underlying archive format does not have the correct information, although the correct number of  
 27594 <blank>-separated fields shall be written.

27595 In **list** mode, standard output shall not be buffered more than a line at a time.

27596 **STDERR**

27597 If **-v** is specified in **read**, **write**, or **copy** modes, *pax* shall write the pathnames it processes to the  
 27598 standard error output using the following format:

27599 "%s\n", <pathname>

27600 These pathnames shall be written as soon as processing is begun on the file or archive member,  
 27601 and shall be flushed to standard error. The trailing <newline>, which shall not be buffered, is  
 27602 written when the file has been read or written.

27603 If the **-s** option is specified, and the replacement string has a trailing 'p', substitutions shall be  
 27604 written to standard error in the following format:

27605 "%sΔ>>Δ%s\n", <original pathname>, <new pathname>

27606 In all operating modes of *pax*, optional messages of unspecified format concerning the input  
 27607 archive format and volume number, the number of files, blocks, volumes, and media parts as  
 27608 well as other diagnostic messages may be written to standard error.

27609 In all formats, for both standard output and standard error, it is unspecified how non-printable  
 27610 characters in pathnames or link names are written.

27611 When *pax* is in **read** mode or **list** mode, using the **-xpax** archive format, and a filename, link  
 27612 name, owner name, or any other field in an extended header record cannot be translated from  
 27613 the **pax** UTF-8 codeset format to the codeset and current locale of the implementation, *pax* shall  
 27614 write a diagnostic message to standard error, shall process the file as described for the **-o**  
 27615 **invalid=**option, and then shall process the next file in the archive.

27616 **OUTPUT FILES**

27617 In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the  
 27618 copied output files shall be the type of the file being copied. In either mode, existing files in the  
 27619 destination hierarchy shall be overwritten only when all permission (**-p**), modification time (**-u**),  
 27620 and invalid-value (**-oinvalid=**) tests allow it.

27621 In **write** mode, the output file named by the **-f** option-argument shall be a file formatted  
 27622 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other  
 27623 implementation-defined format.

27624 **EXTENDED DESCRIPTION**27625 **pax Interchange Format**

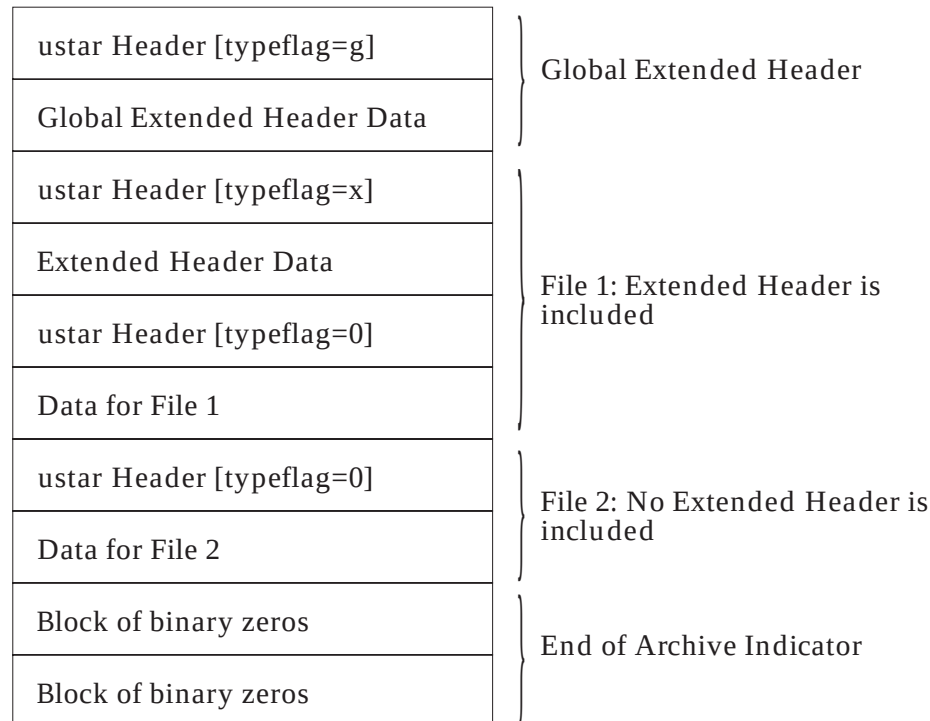
27626 A *pax* archive tape or file produced in the **-xpax** format shall contain a series of blocks. The  
 27627 physical layout of the archive shall be identical to the **ustar** format described in **ustar**  
 27628 **Interchange Format** (on page 717). Each file archived shall be represented by the following  
 27629 sequence:

- 27630 • An optional header block with extended header records. This header block is of the form  
 27631 described in **pax Header Block** (on page 713), with a *typeflag* value of **x** or **g**. The extended  
 27632 header records, described in **pax Extended Header** (on page 714), shall be included as the  
 27633 data for this header block.
- 27634 • A header block that describes the file. Any fields in the preceding optional extended header  
 27635 shall override the associated fields in this header block for this file.
- 27636 • Zero or more blocks that contain the contents of the file.

27637 At the end of the archive file there shall be two 512-byte blocks filled with binary zeros,  
 27638 interpreted as an end-of-archive indicator.

27639 A schematic of an example archive with global extended header records and two actual files is  
 27640 shown in Figure 4-1. In the example, the second file in the archive has no extended header  
 27641 preceding it, presumably because it has no need for extended attributes.

27642



27643

**Figure 4-1** pax Format Archive Example

27644

### **pax Header Block**

27645 The **pax** header block shall be identical to the **ustar** header block described in **ustar Interchange**  
 27646 **Format** (on page 717), except that two additional *typeflag* values are defined:

27647 x Represents extended header records for the following file in the archive (which shall have  
 27648 its own **ustar** header block). The format of these extended header records shall be as  
 27649 described in **pax Extended Header** (on page 714).

27650 g Represents global extended header records for the following files in the archive. The format  
 27651 of these extended header records shall be as described in **pax Extended Header** (on page  
 27652 714). Each value shall affect all subsequent files that do not override that value in their own  
 27653 extended header record and until another global extended header record is reached that  
 27654 provides another value for the same field. The *typeflag g* global headers should not be used  
 27655 with interchange media that could suffer partial data loss in transporting the archive.

27656 For both of these types, the *size* field shall be the size of the extended header records in octets.  
 27657 The other fields in the header block are not meaningful to this version of the *pax* utility.  
 27658 However, if this archive is read by a *pax* utility conforming to the ISO POSIX-2: 1993 standard,  
 27659 the header block fields are used to create a regular file that contains the extended header records  
 27660 as data. Therefore, header block field values should be selected to provide reasonable file access  
 27661 to this regular file.

27662 A further difference from the **ustar** header block is that data blocks for files of *typeflag* 1 (the digit  
 27663 one) (hard link) may be included, which means that the size field may be greater than zero.  
 27664 Archives created by *pax -o linkdata* shall include these data blocks with the hard links.

## 27665 **pax Extended Header**

27666 A **pax** extended header contains values that are inappropriate for the **ustar** header block because  
 27667 of limitations in that format: fields requiring a character encoding other than that described in  
 27668 the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar**  
 27669 header, and fields whose format or length do not fit the requirements of the **ustar** header. The  
 27670 values in an extended header add attributes to the following file (or files; see the description of  
 27671 the *typeflag g* header block) or override values in the following header block(s), as indicated in  
 27672 the following list of keywords.

27673 An extended header shall consist of one or more records, each constructed as follows:

27674 "%d %s=%s\n", <length>, <keyword>, <value>

27675 The extended header records shall be encoded according to the ISO/IEC 10646-1:2000 standard  
 27676 (UTF-8). The <length> field, <blank>, equals sign, and <newline> shown shall be limited to the  
 27677 portable character set, as encoded in UTF-8. The <keyword> and <value> fields can be any UTF-8  
 27678 characters. The <length> field shall be the decimal length of the extended header record in octets,  
 27679 including the trailing <newline>.

27680 The <keyword> field shall be one of the entries from the following list or a keyword provided as  
 27681 an implementation extension. Keywords consisting entirely of lowercase letters, digits, and  
 27682 periods are reserved for future standardization. A keyword shall not include an equals sign. (In  
 27683 the following list, the notations “file(s)” or “block(s)” is used to acknowledge that a keyword  
 27684 affects the following single file after a *typeflag x* extended header, but possibly multiple files after  
 27685 *typeflag g*. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode  
 27686 shall apply only when such a record has not already been provided through the use of the **-o**  
 27687 option. When used in **copy** mode, *pax* shall behave as if an archive had been created with  
 27688 applicable extended header records and then extracted.)

27689 **atime** The file access time for the following file(s), equivalent to the value of the *st\_atime*  
 27690 member of the **stat** structure for a file, as described by the *stat()* function. The  
 27691 access time shall be restored if the process has the appropriate privilege required  
 27692 to do so. The format of the <value> shall be as described in **pax Extended Header**  
 27693 **File Times** (on page 717).

27694 **charset** The name of the character set used to encode the data in the following file(s). The  
 27695 entries in the following table are defined to refer to known standards; additional  
 27696 names may be agreed on between the originator and recipient.

27697  
27698  
27699  
27700  
27701  
27702  
27703  
27704  
27705  
27706  
27707  
27708  
27709  
27710  
27711  
27712  
27713  
27714  
27715

| <value>                 | Formal Standard                 |
|-------------------------|---------------------------------|
| ISO-IRΔ646Δ1990         | ISO / IEC 646: 1990             |
| ISO-IRΔ8859Δ1Δ1998      | ISO / IEC 8859-1: 1998          |
| ISO-IRΔ8859Δ2Δ1999      | ISO / IEC 8859-2: 1999          |
| ISO-IRΔ8859Δ3Δ1999      | ISO / IEC 8859-3: 1999          |
| ISO-IRΔ8859Δ4Δ1998      | ISO / IEC 8859-4: 1998          |
| ISO-IRΔ8859Δ5Δ1999      | ISO / IEC 8859-5: 1999          |
| ISO-IRΔ8859Δ6Δ1999      | ISO / IEC 8859-6: 1999          |
| ISO-IRΔ8859Δ7Δ1987      | ISO / IEC 8859-7: 1987          |
| ISO-IRΔ8859Δ8Δ1999      | ISO / IEC 8859-8: 1999          |
| ISO-IRΔ8859Δ9Δ1999      | ISO / IEC 8859-9: 1999          |
| ISO-IRΔ8859Δ10Δ1998     | ISO / IEC 8859-10: 1998         |
| ISO-IRΔ8859Δ13Δ1998     | ISO / IEC 8859-13: 1998         |
| ISO-IRΔ8859Δ14Δ1998     | ISO / IEC 8859-14: 1998         |
| ISO-IRΔ8859Δ15Δ1999     | ISO / IEC 8859-15: 1999         |
| ISO-IRΔ10646Δ2000       | ISO / IEC 10646: 2000           |
| ISO-IRΔ10646Δ2000ΔUTF-8 | ISO / IEC 10646, UTF-8 encoding |
| BINARY                  | None.                           |

27716  
27717  
27718

The encoding is included in an extended header for information only; when *pax* is used as described in IEEE Std 1003.1-2001, it shall not translate the file data into any other encoding. The **BINARY** entry indicates unencoded binary data.

27719  
27720

When used in **write** or **copy** mode, it is implementation-defined whether *pax* includes a **charset** extended header record for a file.

27721 **comment**  
27722

A series of characters used as a comment. All characters in the <value> field shall be ignored by *pax*.

27723 **ctime**  
27724  
27725  
27726  
27727

The file creation time for the following file(s), equivalent to the value of the *st\_ctime* member of the **stat** structure for a file, as described by the *stat()* function. The creation time shall be restored if the process has the appropriate privilege required to do so. The format of the <value> shall be as described in **pax Extended Header File Times** (on page 717).

27728 **gid**  
27729  
27730  
27731  
27732

The group ID of the group that owns the file, expressed as a decimal number using digits from the ISO / IEC 646: 1991 standard. This record shall override the *gid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *gid* extended header record for each file whose group ID is greater than 2 097 151 (octal 7 777 777).

27733 **gname**  
27734  
27735  
27736  
27737  
27738  
27739  
27740  
27741  
27742

The group of the file(s), formatted as a group name in the group database. This record shall override the *gid* and *gname* fields in the following header block(s), and any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the UTF-8 encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the UTF-8 characters cannot be translated, and if the **-oinvalid=UTF-8** option is not specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **gname** extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.

27743 **linkpath**  
27744  
27745

The pathname of a link being created to another file, of any type, previously archived. This record shall override the *linkname* field in the following **ustar** header block(s). The following **ustar** header block shall determine the type of link created.

|       |                     |                                                                                                                     |
|-------|---------------------|---------------------------------------------------------------------------------------------------------------------|
| 27746 |                     | If <i>typeflag</i> of the following header block is 1, it shall be a hard link. If <i>typeflag</i> is 2, it         |
| 27747 |                     | shall be a symbolic link and the <b>linkpath</b> value shall be the contents of the                                 |
| 27748 |                     | symbolic link. The <i>pax</i> utility shall translate the name of the link (contents of the                         |
| 27749 |                     | symbolic link) from the UTF-8 encoding to the character set appropriate for the                                     |
| 27750 |                     | local file system. When used in <b>write</b> or <b>copy</b> mode, <i>pax</i> shall include a <b>linkpath</b>        |
| 27751 |                     | extended header record for each link whose pathname cannot be represented                                           |
| 27752 |                     | entirely with the members of the portable character set other than NUL.                                             |
| 27753 | <b>mtime</b>        | The file modification time of the following file(s), equivalent to the value of the                                 |
| 27754 |                     | <i>st_mtime</i> member of the <b>stat</b> structure for a file, as described in the <i>stat()</i> function.         |
| 27755 |                     | This record shall override the <i>mtime</i> field in the following header block(s). The                             |
| 27756 |                     | modification time shall be restored if the process has the appropriate privilege                                    |
| 27757 |                     | required to do so. The format of the <value> shall be as described in <b>pax Extended</b>                           |
| 27758 |                     | <b>Header File Times</b> (on page 717).                                                                             |
| 27759 | <b>path</b>         | The pathname of the following file(s). This record shall override the <i>name</i> and                               |
| 27760 |                     | <i>prefix</i> fields in the following header block(s). The <i>pax</i> utility shall translate the                   |
| 27761 |                     | pathname of the file from the UTF-8 encoding to the character set appropriate for                                   |
| 27762 |                     | the local file system.                                                                                              |
| 27763 |                     | When used in <b>write</b> or <b>copy</b> mode, <i>pax</i> shall include a <i>path</i> extended header record        |
| 27764 |                     | for each file whose pathname cannot be represented entirely with the members of                                     |
| 27765 |                     | the portable character set other than NUL.                                                                          |
| 27766 | <b>realtime.any</b> | The keywords prefixed by “realtime.” are reserved for future standardization.                                       |
| 27767 | <b>security.any</b> | The keywords prefixed by “security.” are reserved for future standardization.                                       |
| 27768 | <b>size</b>         | The size of the file in octets, expressed as a decimal number using digits from the                                 |
| 27769 |                     | ISO/IEC 646:1991 standard. This record shall override the <i>size</i> field in the                                  |
| 27770 |                     | following header block(s). When used in <b>write</b> or <b>copy</b> mode, <i>pax</i> shall include a                |
| 27771 |                     | <i>size</i> extended header record for each file with a size value greater than 8 589 934 591                       |
| 27772 |                     | (octal 77 777 777 777).                                                                                             |
| 27773 | <b>uid</b>          | The user ID of the file owner, expressed as a decimal number using digits from the                                  |
| 27774 |                     | ISO/IEC 646:1991 standard. This record shall override the <i>uid</i> field in the                                   |
| 27775 |                     | following header block(s). When used in <b>write</b> or <b>copy</b> mode, <i>pax</i> shall include a                |
| 27776 |                     | <i>uid</i> extended header record for each file whose owner ID is greater than 2 097 151                            |
| 27777 |                     | (octal 7 777 777).                                                                                                  |
| 27778 | <b>uname</b>        | The owner of the following file(s), formatted as a user name in the user database.                                  |
| 27779 |                     | This record shall override the <i>uid</i> and <i>uname</i> fields in the following header block(s),                 |
| 27780 |                     | and any <i>uid</i> extended header record. When used in <b>read</b> , <b>copy</b> , or <b>list</b> mode, <i>pax</i> |
| 27781 |                     | shall translate the name from the UTF-8 encoding in the header record to the                                        |
| 27782 |                     | character set appropriate for the user database on the receiving system. If any of                                  |
| 27783 |                     | the UTF-8 characters cannot be translated, and if the <b>-oinvalid=</b> UTF-8 option is                             |
| 27784 |                     | not specified, the results are implementation-defined. When used in <b>write</b> or <b>copy</b>                     |
| 27785 |                     | mode, <i>pax</i> shall include a <b>uname</b> extended header record for each file whose user                       |
| 27786 |                     | name cannot be represented entirely with the letters and digits of the portable                                     |
| 27787 |                     | character set.                                                                                                      |
| 27788 |                     | If the <value> field is zero length, it shall delete any header block field, previously entered                     |
| 27789 |                     | extended header value, or global extended header value of the same name.                                            |
| 27790 |                     | If a keyword in an extended header record (or in a <b>-o</b> option-argument) overrides or deletes a                |
| 27791 |                     | corresponding field in the <b>ustar</b> header block, <i>pax</i> shall ignore the contents of that header block     |
| 27792 |                     | field.                                                                                                              |



27793 Unlike the **ustar** header block fields, NULs shall not delimit *<value>*s; all characters within the  
 27794 *<value>* field shall be considered data for the field. None of the length limitations of the **ustar**  
 27795 header block fields in Table 4-13 (on page 718) shall apply to the extended header records.

#### 27796 **pax Extended Header Keyword Precedence**

27797 This section describes the precedence in which the various header records and fields and  
 27798 command line options are selected to apply to a file in the archive. When *pax* is used in **read** or  
 27799 **list** modes, it shall determine a file attribute in the following sequence:

- 27800 1. If **-odelete=keyword-prefix** is used, the affected attributes shall be determined from step 7.,  
 27801 if applicable, or ignored otherwise.
- 27802 2. If **-okeyword:=** is used, the affected attributes shall be ignored.
- 27803 3. If **-okeyword:=value** is used, the affected attribute shall be assigned the value.
- 27804 4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the  
 27805 *<value>*. When extended header records conflict, the last one given in the header shall take  
 27806 precedence.
- 27807 5. If **-okeyword=value** is used, the affected attribute shall be assigned the value.
- 27808 6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be  
 27809 assigned the *<value>*. When global extended header records conflict, the last one given in  
 27810 the global header shall take precedence.
- 27811 7. Otherwise, the attribute shall be determined from the **ustar** header block.

#### 27812 **pax Extended Header File Times**

27813 The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's  
 27814 modification time cannot be represented exactly in the **ustar** header logical record described in  
 27815 **ustar Interchange Format**. This can occur if the time is out of **ustar** range, or if the file system of  
 27816 the underlying implementation supports non-integer time granularities and the time is not an  
 27817 integer. All of these time records shall be formatted as a decimal representation of the time in  
 27818 seconds since the Epoch. If a period ( ' . ' ) decimal point character is present, the digits to the  
 27819 right of the point shall represent the units of a subsecond timing granularity, where the first digit  
 27820 is tenths of a second and each subsequent digit is a tenth of the previous digit. In **read** or **copy**  
 27821 mode, the *pax* utility shall truncate the time of a file to the greatest value that is not greater than  
 27822 the input header file time. In **write** or **copy** mode, the *pax* utility shall output a time exactly if it  
 27823 can be represented exactly as a decimal number, and otherwise shall generate only enough digits  
 27824 so that the same time shall be recovered if the file is extracted on a system whose underlying  
 27825 implementation supports the same time granularity.

#### 27826 **ustar Interchange Format**

27827 A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a  
 27828 fixed-size logical record of 512 octets (see below). Although this format may be thought of as  
 27829 being stored on 9-track industry-standard 12.7 mm (0.5 in) magnetic tape, other types of  
 27830 transportable media are not excluded. Each file archived shall be represented by a header logical  
 27831 record that describes the file, followed by zero or more logical records that give the contents of  
 27832 the file. At the end of the archive file there shall be two 512-octet logical records filled with  
 27833 binary zeros, interpreted as an end-of-archive indicator.

27834 The logical records may be grouped for physical I/O operations, as described under the  
 27835 **-bblocksize** and **-x ustar** options. Each group of logical records may be written with a single  
 27836 operation equivalent to the *write()* function. On magnetic tape, the result of this write shall be a

single tape physical block. The last physical block shall always be the full size, so logical records after the two zero logical records may contain undefined data.

The header logical record shall be structured as shown in the following table. All lengths and offsets are in decimal.

**Table 4-13** ustar Header Block

| Field Name      | Octet Offset | Length (in Octets) |
|-----------------|--------------|--------------------|
| <i>name</i>     | 0            | 100                |
| <i>mode</i>     | 100          | 8                  |
| <i>uid</i>      | 108          | 8                  |
| <i>gid</i>      | 116          | 8                  |
| <i>size</i>     | 124          | 12                 |
| <i>mtime</i>    | 136          | 12                 |
| <i>chksum</i>   | 148          | 8                  |
| <i>typeflag</i> | 156          | 1                  |
| <i>linkname</i> | 157          | 100                |
| <i>magic</i>    | 257          | 6                  |
| <i>version</i>  | 263          | 2                  |
| <i>uname</i>    | 265          | 32                 |
| <i>gname</i>    | 297          | 32                 |
| <i>devmajor</i> | 329          | 8                  |
| <i>devminor</i> | 337          | 8                  |
| <i>prefix</i>   | 345          | 155                |

All characters in the header logical record shall be represented in the coded character set of the ISO/IEC 646:1991 standard. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of slash and the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes.

However, the *pax* utility shall never create filenames on the local system that cannot be accessed via the procedures described in IEEE Std 1003.1-2001. If a filename is found on the medium that would create an invalid filename, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

Each field within the header logical record is contiguous; that is, there is no padding used. Each character on the archive medium shall be stored contiguously.

The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character. The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all characters in the array contain non-NUL characters including the last character. The *version* field is two octets containing the characters "00" (zero-zero). The *typeflag* contains a single character. All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up to the first NUL character), a slash character, and *name*; otherwise, *name* is used alone. In either case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it shall be ignored. In this manner, pathnames of at most 256 characters can be supported. If a pathname

27884 does not fit in the space provided, *pax* shall notify the user of the error, and shall not store any  
27885 part of the file—header or data—on the medium.

27886 The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a  
27887 *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall  
27888 notify the user of the error, and shall not attempt to store the link on the medium.

27889 The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit  
27890 representation. The encoded bits shall represent the following values:

27891 **Table 4-14** *ustar mode* Field

| 27892 | Bit Value | IEEE Std 1003.1-2001 Bit | Description                                     |
|-------|-----------|--------------------------|-------------------------------------------------|
| 27893 | 04 000    | S_ISUID                  | Set UID on execution.                           |
| 27894 | 02 000    | S_ISGID                  | Set GID on execution.                           |
| 27895 | 01 000    | <reserved>               | Reserved for future standardization.            |
| 27896 | 00 400    | S_IRUSR                  | Read permission for file owner class.           |
| 27897 | 00 200    | S_IWUSR                  | Write permission for file owner class.          |
| 27898 | 00 100    | S_IXUSR                  | Execute/search permission for file owner class. |
| 27899 | 00 040    | S_IRGRP                  | Read permission for file group class.           |
| 27900 | 00 020    | S_IWGRP                  | Write permission for file group class.          |
| 27901 | 00 010    | S_IXGRP                  | Execute/search permission for file group class. |
| 27902 | 00 004    | S_IROTH                  | Read permission for file other class.           |
| 27903 | 00 002    | S_IWOTH                  | Write permission for file other class.          |
| 27904 | 00 001    | S_IXOTH                  | Execute/search permission for file other class. |

27905 When appropriate privilege is required to set one of these mode bits, and the user restoring the  
27906 files from the archive does not have the appropriate privilege, the mode bits for which the user  
27907 does not have appropriate privilege shall be ignored. Some of the mode bits in the archive  
27908 format are not mentioned elsewhere in this volume of IEEE Std 1003.1-2001. If the  
27909 implementation does not support those bits, they may be ignored.

27910 The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

27911 The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type  
27912 1 (a link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is set to  
27913 specify a file of type 5 (directory), the *size* field shall be interpreted as described under the  
27914 definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag*  
27915 field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size*  
27916 field is unspecified by this volume of IEEE Std 1003.1-2001, and no data logical records shall be  
27917 stored on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If  
27918 the *typeflag* field is set to any other value, the number of logical records written following the  
27919 header shall be  $(size+511)/512$ , ignoring any fraction in the result of the division.

27920 The *mtime* field shall be the modification time of the file at the time it was archived. It is the  
27921 ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained  
27922 from the *stat()* function.

27923 The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of  
27924 the simple sum of all octets in the header logical record. Each octet in the header shall be treated  
27925 as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the  
27926 precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is  
27927 treated as if it were all spaces.

27928 The *typeflag* field specifies the type of file archived. If a particular implementation does not  
27929 recognize the type, or the user does not have appropriate privilege to create that type, the file

|       |       |                                                                                                                      |
|-------|-------|----------------------------------------------------------------------------------------------------------------------|
| 27930 |       | shall be extracted as if it were a regular file if the file type is defined to have a meaning for the                |
| 27931 |       | <i>size</i> field that could cause data logical records to be written on the medium (see the previous                |
| 27932 |       | description for <i>size</i> ). If conversion to a regular file occurs, the <i>pax</i> utility shall produce an error |
| 27933 |       | indicating that the conversion took place. All of the <i>typeflag</i> fields shall be coded in the                   |
| 27934 |       | ISO / IEC 646: 1991 standard IRV:                                                                                    |
| 27935 | 0     | Represents a regular file. For backwards-compatibility, a <i>typeflag</i> value of binary zero                       |
| 27936 |       | (' \0 ') should be recognized as meaning a regular file when extracting files from the                               |
| 27937 |       | archive. Archives written with this version of the archive file format create regular files                          |
| 27938 |       | with a <i>typeflag</i> value of the ISO / IEC 646: 1991 standard IRV ' 0 '.                                          |
| 27939 | 1     | Represents a file linked to another file, of any type, previously archived. Such files are                           |
| 27940 |       | identified by each file having the same device and file serial number. The linked-to                                 |
| 27941 |       | name is specified in the <i>linkname</i> field with a NUL-character terminator if it is less than                    |
| 27942 |       | 100 octets in length.                                                                                                |
| 27943 | 2     | Represents a symbolic link. The contents of the symbolic link shall be stored in the                                 |
| 27944 |       | <i>linkname</i> field.                                                                                               |
| 27945 | 3 , 4 | Represent character special files and block special files respectively. In this case the                             |
| 27946 |       | <i>devmajor</i> and <i>devminor</i> fields shall contain information defining the device, the format                 |
| 27947 |       | of which is unspecified by this volume of IEEE Std 1003.1-2001. Implementations may                                  |
| 27948 |       | map the device specifications to their own local specification or may ignore the entry.                              |
| 27949 | 5     | Specifies a directory or subdirectory. On systems where disk allocation is performed on                              |
| 27950 |       | a directory basis, the <i>size</i> field shall contain the maximum number of octets (which may                       |
| 27951 |       | be rounded to the nearest disk block allocation unit) that the directory may hold. A <i>size</i>                     |
| 27952 |       | field of zero indicates no such limiting. Systems that do not support limiting in this                               |
| 27953 |       | manner should ignore the <i>size</i> field.                                                                          |
| 27954 | 6     | Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence                         |
| 27955 |       | of this file and not its contents.                                                                                   |
| 27956 | 7     | Reserved to represent a file to which an implementation has associated some high-                                    |
| 27957 |       | performance attribute. Implementations without such extensions should treat this file                                |
| 27958 |       | as a regular file (type 0).                                                                                          |
| 27959 | A - Z | The letters ' A ' to ' Z ', inclusive, are reserved for custom implementations. All other                            |
| 27960 |       | values are reserved for future versions of IEEE Std 1003.1-2001.                                                     |
| 27961 |       | Attempts to archive a socket using <b>ustar</b> interchange format shall produce a diagnostic message.               |
| 27962 |       | Handling of other file types is implementation-defined.                                                              |
| 27963 |       | The <i>magic</i> field is the specification that this archive was output in this archive format. If this field       |
| 27964 |       | contains <b>ustar</b> (the five characters from the ISO / IEC 646: 1991 standard IRV shown followed by               |
| 27965 |       | NUL), the <i>uname</i> and <i>gname</i> fields shall contain the ISO / IEC 646: 1991 standard IRV                    |
| 27966 |       | representation of the owner and group of the file, respectively (truncated to fit, if necessary).                    |
| 27967 |       | When the file is restored by a privileged, protection-preserving version of the utility, the user                    |
| 27968 |       | and group databases shall be scanned for these names. If found, the user and group IDs                               |
| 27969 |       | contained within these files shall be used rather than the values contained within the <i>uid</i> and <i>gid</i>     |
| 27970 |       | fields.                                                                                                              |

**cpio Interchange Format**

The octet-oriented **cpio** archive format shall be a series of entries, each comprising a header that describes the file, the name of the file, and then the contents of the file.

An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used only to make physical I/O more efficient. The last group of blocks shall always be at the full size.

For the octet-oriented **cpio** archive format, the individual entry information shall be in the order indicated and described by the following table; see also the **<cpio.h>** header.

**Table 4-15** Octet-Oriented cpio Archive Entry

| Header Field Name    | Length (in Octets) | Interpreted as  |
|----------------------|--------------------|-----------------|
| <i>c_magic</i>       | 6                  | Octal number    |
| <i>c_dev</i>         | 6                  | Octal number    |
| <i>c_ino</i>         | 6                  | Octal number    |
| <i>c_mode</i>        | 6                  | Octal number    |
| <i>c_uid</i>         | 6                  | Octal number    |
| <i>c_gid</i>         | 6                  | Octal number    |
| <i>c_nlink</i>       | 6                  | Octal number    |
| <i>c_rdev</i>        | 6                  | Octal number    |
| <i>c_mtime</i>       | 11                 | Octal number    |
| <i>c_namesize</i>    | 6                  | Octal number    |
| <i>c_filesize</i>    | 11                 | Octal number    |
| Filename Field Name  | Length             | Interpreted as  |
| <i>c_name</i>        | <i>c_namesize</i>  | Pathname string |
| File Data Field Name | Length             | Interpreted as  |
| <i>c_filedata</i>    | <i>c_filesize</i>  | Data            |

**cpio Header**

For each file in the archive, a header as defined previously shall be written. The information in the header fields is written as streams of the ISO/IEC 646:1991 standard characters interpreted as octal numbers. The octal numbers shall be extended to the necessary length by appending the ISO/IEC 646:1991 standard IRV zeros at the most-significant-digit end of the number; the result is written to the most-significant digit of the stream of octets first. The fields shall be interpreted as follows:

*c\_magic* Identify the archive as being a transportable archive by containing the identifying value "070707".

*c\_dev, c\_ino* Contains values that uniquely identify the file within the archive (that is, no files contain the same pair of *c\_dev* and *c\_ino* values unless they are links to the same file). The values shall be determined in an unspecified manner.

*c\_mode* Contains the file type and access permissions as defined in the following table.

28009

**Table 4-16** Values for cpio c\_mode Field

28010

28011

28012

28013

28014

28015

28016

28017

28018

28019

28020

28021

28022

28023

28024

28025

28026

28027

28028

28029

28030

28031

| File Permissions Name | Value    | Indicates              |
|-----------------------|----------|------------------------|
| C_IRUSR               | 000 400  | Read by owner          |
| C_IWUSR               | 000 200  | Write by owner         |
| C_IXUSR               | 000 100  | Execute by owner       |
| C_IRGRP               | 000 040  | Read by group          |
| C_IWGRP               | 000 020  | Write by group         |
| C_IXGRP               | 000 010  | Execute by group       |
| C_IROTH               | 000 004  | Read by others         |
| C_IWOTH               | 000 002  | Write by others        |
| C_IXOTH               | 000 001  | Execute by others      |
| C_ISUID               | 004 000  | Set <i>uid</i>         |
| C_ISGID               | 002 000  | Set <i>gid</i>         |
| C_ISVTX               | 001 000  | Reserved               |
| File Type Name        | Value    | Indicates              |
| C_ISDIR               | 040 000  | Directory              |
| C_ISFIFO              | 010 000  | FIFO                   |
| C_ISREG               | 0100 000 | Regular file           |
| C_ISLNK               | 0120 000 | Symbolic link          |
| C_ISBLK               | 060 000  | Block special file     |
| C_ISCHR               | 020 000  | Character special file |
| C_ISSOCK              | 0140 000 | Socket                 |
| C_ISCTG               | 0110 000 | Reserved               |

28032

28033

28034

28035

28036

Directories, FIFOs, symbolic links, and regular files shall be supported on a system conforming to this volume of IEEE Std 1003.1-2001; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

28037

*c\_uid*

Contains the user ID of the owner.

28038

*c\_gid*

Contains the group ID of the group.

28039

*c\_nlink*

Contains the number of links referencing the file at the time the archive was created.

28040

28041

*c\_rdev*

Contains implementation-defined information for character or block special files.

28042

*c\_mtime*

Contains the latest time of modification of the file at the time the archive was created.

28043

28044

*c\_namesize*

Contains the length of the pathname, including the terminating NUL character.

28045

*c\_filesize*

Contains the length of the file in octets. This shall be the length of the data section following the header structure.

28046

28047 **cpio Filename**

28048 The *c\_name* field shall contain the pathname of the file. The length of this field in octets is the  
 28049 value of *c\_namesize*.

28050 If a filename is found on the medium that would create an invalid pathname, it is  
 28051 implementation-defined whether the data from the file is stored on the file hierarchy and under  
 28052 what name it is stored.

28053 All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum  
 28054 portability between implementations, names should be selected from characters represented by  
 28055 the portable filename character set as octets with the most significant bit zero. If an  
 28056 implementation supports the use of characters outside the portable filename character set in  
 28057 names for files, users, and groups, one or more implementation-defined encodings of these  
 28058 characters shall be provided for interchange purposes. However, the *pax* utility shall never  
 28059 create filenames on the local system that cannot be accessed via the procedures described  
 28060 previously in this volume of IEEE Std 1003.1-2001. If a filename is found on the medium that  
 28061 would create an invalid filename, it is implementation-defined whether the data from the file is  
 28062 stored on the local file system and under what name it is stored. The *pax* utility may choose to  
 28063 ignore these files as long as it produces an error indicating that the file is being ignored.

28064 **cpio File Data**

28065 Following *c\_name*, there shall be *c\_filesize* octets of data. Interpretation of such data occurs in a  
 28066 manner dependent on the file. If *c\_filesize* is zero, no data shall be contained in *c\_filedata*.

28067 When restoring from an archive:

- 28068 • If the user does not have the appropriate privilege to create a file of the specified type, *pax*  
 28069 shall ignore the entry and write an error message to standard error.
- 28070 • Only regular files have data to be restored. Presuming a regular file meets any selection  
 28071 criteria that might be imposed on the format-reading utility by the user, such data shall be  
 28072 restored.
- 28073 • If a user does not have appropriate privilege to set a particular mode flag, the flag shall be  
 28074 ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this  
 28075 volume of IEEE Std 1003.1-2001. If the implementation does not support those flags, they  
 28076 may be ignored.

28077 **cpio Special Entries**

28078 FIFO special files, directories, and the trailer shall be recorded with *c\_filesize* equal to zero. For  
 28079 other special files, *c\_filesize* is unspecified by this volume of IEEE Std 1003.1-2001. The header for  
 28080 the next file entry in the archive shall be written directly after the last octet of the file entry  
 28081 preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive;  
 28082 the contents of octets in the last block of the archive following such a header are undefined.

28083 **EXIT STATUS**

28084 The following exit values shall be returned:

- 28085 0 All files were processed successfully.
- 28086 >0 An error occurred.

28087 **CONSEQUENCES OF ERRORS**

28088 If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an  
 28089 archive, or cannot preserve the user ID, group ID, or file mode when the **-p** option is specified, a  
 28090 diagnostic message shall be written to standard error and a non-zero exit status shall be  
 28091 returned, but processing shall continue. In the case where *pax* cannot create a link to a file, *pax*  
 28092 shall not, by default, create a second copy of the file.

28093 If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may  
 28094 have only partially extracted the file or (if the **-n** option was not specified) may have extracted a  
 28095 file of the same name as that specified by the user, but which is not the file the user wanted.  
 28096 Additionally, the file modes of extracted directories may have additional bits from the S\_IRWXU  
 28097 mask set as well as incorrect modification and access times.

28098 **APPLICATION USAGE**

28099 The **-p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio*  
 28100 implementations. In particular, the two utilities use **-m** in diametrically opposed ways. The **-p**  
 28101 option also provides a consistent means of extending the ways in which future file attributes can  
 28102 be addressed, such as for enhanced security systems or high-performance files. Although it may  
 28103 seem complex, there are really two modes that are most commonly used:

28104 **-p e** “Preserve everything”. This would be used by the historical superuser, someone with  
 28105 all the appropriate privileges, to preserve all aspects of the files as they are recorded in  
 28106 the archive. The **e** flag is the sum of **o** and **p**, and other implementation-defined  
 28107 attributes.

28108 **-p p** “Preserve” the file mode bits. This would be used by the user with regular privileges  
 28109 who wished to preserve aspects of the file other than the ownership. The file times are  
 28110 preserved by default, but two other flags are offered to disable these and use the time  
 28111 of extraction.

28112 The one pathname per line format of standard input precludes pathnames containing  
 28113 <newline>s. Although such pathnames violate the portable filename guidelines, they may exist  
 28114 and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from  
 28115 historical archive programs. The problem can be avoided by listing filename arguments on the  
 28116 command line instead of on standard input.

28117 It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this  
 28118 volume of IEEE Std 1003.1-2001. Specifically, creating files of type block special or character  
 28119 special, restoring file access times unless the files are owned by the user (the **-t** option), or  
 28120 preserving file owner, group, and mode (the **-p** option) all probably require appropriate  
 28121 privileges.

28122 In **read** mode, implementations are permitted to overwrite files when the archive has multiple  
 28123 members with the same name. This may fail if permissions on the first version of the file do not  
 28124 permit it to be overwritten.

28125 The **cpio** and **ustar** formats can only support files up to 8 589 934 592 bytes ( $8 * 2^{30}$ ) in size.

28126 **EXAMPLES**

28127 The following command:

28128 `pax -w -f /dev/rmt/1m .`

28129 copies the contents of the current directory to tape drive 1, medium density (assuming historical  
 28130 System V device naming procedures—the historical BSD device name would be **/dev/rmt9**).

28131 The following commands:



```

28132 mkdir newdir
28133 pax -rw olddir newdir
28134 copy the olddir directory hierarchy to newdir.
28135 pax -r -s ',^/*usr/*,, ' -f a.pax
28136 reads the archive a.pax, with all files rooted in /usr in the archive extracted relative to the current
28137 directory.
28138 Using the option:
28139 -o listopt="%M %(atime)T %(size)D %(name)s"
28140 overrides the default output description in Standard Output and instead writes:
28141 -rw-rw--- Jan 12 15:53 1492 /usr/foo/bar
28142 Using the options:
28143 -o listopt='%L\t%(size)D\n%.7' \
28144 -o listopt='(name)s\n%(ctime)T\n%T'
28145 overrides the default output description in Standard Output and instead writes:
28146 /usr/foo/bar -> /tmp 1492
28147 /usr/fo
28148 Jan 12 1991
28149 Jan 31 15:53
28150 RATIONALE
28151 The pax utility was new for the ISO POSIX-2:1993 standard. It represents a peaceful compromise
28152 between advocates of the historical tar and cpio utilities.
28153 A fundamental difference between cpio and tar was in the way directories were treated. The cpio
28154 utility did not treat directories differently from other files, and to select a directory and its
28155 contents required that each file in the hierarchy be explicitly specified. For tar, a directory
28156 matched every file in the file hierarchy it rooted.
28157 The pax utility offers both interfaces; by default, directories map into the file hierarchy they root.
28158 The -d option causes pax to skip any file not explicitly referenced, as cpio historically did. The tar
28159 -style behavior was chosen as the default because it was believed that this was the more
28160 common usage and because tar is the more commonly available interface, as it was historically
28161 provided on both System V and BSD implementations.
28162 The data interchange format specification in this volume of IEEE Std 1003.1-2001 requires that
28163 processes with “appropriate privileges” shall always restore the ownership and permissions of
28164 extracted files exactly as archived. If viewed from the historic equivalence between superuser
28165 and “appropriate privileges”, there are two problems with this requirement. First, users running
28166 as superusers may unknowingly set dangerous permissions on extracted files. Second, it is
28167 needlessly limiting, in that superusers cannot extract files and own them as superuser unless the
28168 archive was created by the superuser. (It should be noted that restoration of ownerships and
28169 permissions for the superuser, by default, is historical practice in cpio, but not in tar.) In order to
28170 avoid these two problems, the pax specification has an additional “privilege” mechanism, the -p
28171 option. Only a pax invocation with the privileges needed, and which has the -p option set using
28172 the e specification character, has the “appropriate privilege” to restore full ownership and
28173 permission information.
28174 Note also that this volume of IEEE Std 1003.1-2001 requires that the file ownership and access
28175 permissions shall be set, on extraction, in the same fashion as the creat() function when provided

```

28176 with the mode stored in the archive. This means that the file creation mask of the user is applied  
28177 to the file permissions.

28178 Users should note that directories may be created by *pax* while extracting files with permissions  
28179 that are different from those that existed at the time the archive was created. When extracting  
28180 sensitive information into a directory hierarchy that no longer exists, users are encouraged to set  
28181 their file creation mask appropriately to protect these files during extraction.

28182 The table of contents output is written to standard output to facilitate pipeline processing.

28183 An early proposal had hard links displaying for all pathnames. This was removed because it  
28184 complicates the output of the case where **-v** is not specified and does not match historical *cpio*  
28185 usage. The hard-link information is available in the **-v** display.

28186 The description of the **-l** option allows implementations to make hard links to symbolic links.  
28187 IEEE Std 1003.1-2001 does not specify any way to create a hard link to a symbolic link, but many  
28188 implementations provide this capability as an extension. If there are hard links to symbolic links  
28189 when an archive is created, the implementation is required to archive the hard link in the archive  
28190 (unless **-H** or **-L** is specified). When in **read** mode and in **copy** mode, implementations  
28191 supporting hard links to symbolic links should use them when appropriate.

28192 The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that  
28193 have been brought along from historical usage. For example, there are restrictions on the length  
28194 of pathnames stored in the archive. When *pax* is used in **copy(-rw)** mode (copying directory  
28195 hierarchies), the ability to use extensions from the **-xpax** format overcomes these restrictions.

28196 The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard  
28197 block-size values for *cpio*, set when the **-B** option is specified. (The other default block-size value  
28198 for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240  
28199 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The  
28200 maximum block size of 32 256 bytes ( $2^{15}$ –512 bytes) is the largest multiple of 512 bytes that fits  
28201 into a signed 16-bit tape controller transfer register. There are known limitations in some  
28202 historical systems that would prevent larger blocks from being accepted. Historical values were  
28203 chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate  
28204 archives. Also, default block sizes for any file type other than character special file has been  
28205 deleted from this volume of IEEE Std 1003.1-2001 as unimportant and not likely to affect the  
28206 structure of the resulting archive.

28207 Implementations are permitted to modify the block-size value based on the archive format or  
28208 the device to which the archive is being written. This is to provide implementations with the  
28209 opportunity to take advantage of special types of devices, and it should not be used without a  
28210 great deal of consideration as it almost certainly decreases archive portability.

28211 The intended use of the **-n** option was to permit extraction of one or more files from the archive  
28212 without processing the entire archive. This was viewed by the standard developers as offering  
28213 significant performance advantages over historical implementations. The **-n** option in early  
28214 proposals had three effects; the first was to cause special characters in patterns to not be treated  
28215 specially. The second was to cause only the first file that matched a pattern to be extracted. The  
28216 third was to cause *pax* to write a diagnostic message to standard error when no file was found  
28217 matching a specified pattern. Only the second behavior is retained by this volume of  
28218 IEEE Std 1003.1-2001, for many reasons. First, it is in general not acceptable for a single option to  
28219 have multiple effects. Second, the ability to make pattern matching characters act as normal  
28220 characters is useful for parts of *pax* other than file extraction. Third, a finer degree of control over  
28221 the special characters is useful because users may wish to normalize only a single special  
28222 character in a single filename. Fourth, given a more general escape mechanism, the previous  
28223 behavior of the **-n** option can be easily obtained using the **-s** option or a *sed* script. Finally,

28224 writing a diagnostic message when a pattern specified by the user is unmatched by any file is  
28225 useful behavior in all cases.

28226 In this version, the **-n** was removed from the **copy** mode synopsis of *pax*; it is inapplicable  
28227 because there are no pattern operands specified in this mode.

28228 There is another method than *pax* for copying subtrees in IEEE Std 1003.1-2001 described as part  
28229 of the *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive  
28230 interface, while *pax* offers a finer granularity of control. Each provides additional functionality to  
28231 the other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not.  
28232 It is the intention of the standard developers that the results be similar (using appropriate option  
28233 combinations in both utilities). The results are not required to be identical; there seemed  
28234 insufficient gain to applications to balance the difficulty of implementations having to guarantee  
28235 that the results would be exactly identical.

28236 A single archive may span more than one file. It is suggested that implementations provide  
28237 informative messages to the user on standard error whenever the archive file is changed.

28238 The **-d** option (do not create intermediate directories not listed in the archive) found in early  
28239 proposals was originally provided as a complement to the historic **-d** option of *cpio*. It has been  
28240 deleted.

28241 The **-s** option in early proposals specified a subset of the substitution command from the *ed*  
28242 utility. As there was no reason for only a subset to be supported, the **-s** option is now  
28243 compatible with the current *ed* specification. Since the delimiter can be any non-null character,  
28244 the following usage with single spaces is valid:

28245 `pax -s " foo bar " ...`

28246 The **-t** description is worded so as to note that this may cause the access time update caused by  
28247 some other activity (which occurs while the file is being read) to be overwritten.

28248 The default behavior of *pax* with regard to file modification times is the same as historical  
28249 implementations of *tar*. It is not the historical behavior of *cpio*.

28250 Because the **-i** option uses **/dev/tty**, utilities without a controlling terminal are not able to use  
28251 this option.

28252 The **-y** option, found in early proposals, has been deleted because a line containing a single  
28253 period for the **-i** option has equivalent functionality. The special lines for the **-i** option (a single  
28254 period and the empty line) are historical practice in *cpio*.

28255 In early drafts, a **-echarmap** option was included to increase portability of files between systems  
28256 using different coded character sets. This option was omitted because it was apparent that  
28257 consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate  
28258 substitute.

28259 The **-k** option was added to address international concerns about the dangers involved in the  
28260 character set transformations of **-e** (if the target character set were different from the source, the  
28261 filenames might be transformed into names matching existing files) and also was made more  
28262 general to protect files transferred between file systems with different {NAME\_MAX} values  
28263 (truncating a filename on a smaller system might also inadvertently overwrite existing files). As  
28264 stated, it prevents any overwriting, even if the target file is older than the source. This version  
28265 adds more granularity of options to solve this problem by introducing the **-oinvalid=** option—  
28266 specifically the UTF-8 action. (Note that an existing file that is named with a UTF-8 encoding is  
28267 still subject to overwriting in this case. The **-k** option closes that loophole.)

28268 Some of the file characteristics referenced in this volume of IEEE Std 1003.1-2001 might not be  
28269 supported by some archive formats. For example, neither the **tar** nor **cpio** formats contain the

28270 file access time. For this reason, the `e` specification character has been provided, intended to  
 28271 cause all file characteristics specified in the archive to be retained.

28272 It is required that extracted directories, by default, have their access and modification times and  
 28273 permissions set to the values specified in the archive. This has obvious problems in that the  
 28274 directories are almost certainly modified after being extracted and that directory permissions  
 28275 may not permit file creation. One possible solution is to create directories with the mode  
 28276 specified in the archive, as modified by the `umask` of the user, with sufficient permissions to  
 28277 allow file creation. After all files have been extracted, *pax* would then reset the access and  
 28278 modification times and permissions as necessary.

28279 The list-mode formatting description borrows heavily from the one defined by the *printf* utility.  
 28280 However, since there is no separate operand list to get conversion arguments, the format was  
 28281 extended to allow specifying the name of the conversion argument as part of the conversion  
 28282 specification.

28283 The `T` conversion specifier allows time fields to be displayed in any of the date formats. Unlike  
 28284 the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past.  
 28285 This makes parsing the output more predictable.

28286 The `D` conversion specifier handles the ability to display the major/minor or file size, as with *ls*,  
 28287 by using `%-8 (size)D`.

28288 The `L` conversion specifier handles the *ls* display for symbolic links.

28289 Conversion specifiers were added to generate existing known types used for *ls*.

## 28290 **pax Interchange Format**

28291 The new POSIX data interchange format was developed primarily to satisfy international  
 28292 concerns that the **ustar** and **cpio** formats did not provide for file, user, and group names encoded  
 28293 in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers  
 28294 realized that this new POSIX data interchange format should be very extensible because there  
 28295 were other requirements they foresaw in the near future:

- 28296 • Support international character encodings and locale information
- 28297 • Support security information (ACLs, and so on)
- 28298 • Support future file types, such as realtime or contiguous files
- 28299 • Include data areas for implementation use
- 28300 • Support systems with words larger than 32 bits and timers with subsecond granularity

28301 The following were not goals for this format because these are better handled by separate  
 28302 utilities or are inappropriate for a portable format:

- 28303 • Encryption
- 28304 • Compression
- 28305 • Data translation between locales and codesets
- 28306 • *inode* storage

28307 The format chosen to support the goals is an extension of the **ustar** format. Of the two formats  
 28308 previously available, only the **ustar** format was selected for extensions because:

- 28309 • It was easier to extend in an upwards-compatible way. It offered version flags and header  
 28310 block type fields with room for future standardization. The **cpio** format, while possessing a  
 28311 more flexible file naming methodology, could not be extended without breaking some

28312           theoretical implementation or using a dummy filename that could be a legitimate filename.

28313           • Industry experience since the original “*tar wars*” fought in developing the ISO POSIX-1  
28314           standard has clearly been in favor of the **ustar** format, which is generally the default output  
28315           format selected for *pax* implementations on new systems.

28316           The new format was designed with one additional goal in mind: reasonable behavior when an  
28317           older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated  
28318           that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this  
28319           allowed the format to include all the extended information in a pseudo-regular file that preceded  
28320           each real file. An option is given that allows the archive creator to set up reasonable names for  
28321           these files on the older systems. Also, the normative text suggests that reasonable file access  
28322           values be used for this **ustar** header block. Making these header files inaccessible for convenient  
28323           reading and deleting would not be reasonable. File permissions of 600 or 700 are suggested.

28324           The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format  
28325           rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous  
28326           version of *pax*), mandated the behavior of the format-reading utility when it encountered an  
28327           unknown *typeflag*, but was silent about the other two fields.

28328           Early proposals of the first revision to IEEE Std 1003.1-2001 contained a proposed archive format  
28329           that was based on compatibility with the standard for tape files (ISO 1001, similar to the format  
28330           used historically on many mainframes and minicomputers). This format was overly complex  
28331           and required considerable overhead in volume and header records. Furthermore, the standard  
28332           developers felt that it would not be acceptable to the community of POSIX developers, so it was  
28333           later changed to be a format more closely related to historical practice on POSIX systems.

28334           The prefix and name split of pathnames in **ustar** was replaced by the single path extended  
28335           header record for simplicity.

28336           The concept of a global extended header (*typeflag g*) was controversial. If this were applied to an  
28337           archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape  
28338           could be a serious problem; a utility attempting to extract as many files as possible from a  
28339           damaged archive could lose a large percentage of file header information in this case. However,  
28340           if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers  
28341           considerable potential size reductions by eliminating redundant information. Thus, the text  
28342           warns against using the global method for unreliable media and provides a method for  
28343           implanting global information in the extended header for each file, rather than in the *typeflag g*  
28344           records.

28345           No facility for data translation or filtering on a per-file basis is included because the standard  
28346           developers could not invent an interface that would allow this in an efficient manner. If a filter,  
28347           such as encryption or compression, is to be applied to all the files, it is more efficient to apply the  
28348           filter to the entire archive as a single file. The standard developers considered interfaces that  
28349           would invoke a shell script for each file going into or out of the archive, but the system overhead  
28350           in this approach was considered to be too high.

28351           One such approach would be to have **filter=** records that give a pathname for an executable.  
28352           When the program is invoked, the file and archive would be open for standard input/output  
28353           and all the header fields would be available as environment variables or command-line  
28354           arguments. The standard developers did discuss such schemes, but they were omitted from  
28355           IEEE Std 1003.1-2001 due to concerns about excessive overhead. Also, the program itself would  
28356           need to be in the archive if it were to be used portably.

28357           There is currently no portable means of identifying the character set(s) used for a file in the file  
28358           system. Therefore, *pax* has not been given a mechanism to generate charset records  
28359           automatically. The only portable means of doing this is for the user to write the archive using the

28360 **-ocharset=string** command line option. This assumes that all of the files in the archive use the  
 28361 same encoding. The “implementation-defined” text is included to allow for a system that can  
 28362 identify the encodings used for each of its files.

28363 The table of standards that accompanies the charset record description is acknowledged to be  
 28364 very limited. Only a limited number of character set standards is reasonable for maximal  
 28365 interchange. Any character set is, of course, possible by prior agreement. It was suggested that  
 28366 EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal  
 28367 standards, and then only those with reasonably large followings, can be included here, simply as  
 28368 a matter of practicality. The <value>s represent names of officially registered character sets in the  
 28369 format required by the ISO 2375:1985 standard.

28370 The normal comma or <blank>-separated list rules are not followed in the case of keyword  
 28371 options to allow ease of argument parsing for *getopts*.

28372 Further information on character encodings is in **pax Archive Character Set Encoding/Decoding**  
 28373 (on page 732).

28374 The standard developers have reserved keyword name space for vendor extensions. It is  
 28375 suggested that the format to be used is:

28376 *VENDOR.keyword*

28377 where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further  
 28378 suggested that the keyword following the period be named differently than any of the standard  
 28379 keywords so that it could be used for future standardization, if appropriate, by omitting the  
 28380 *VENDOR* prefix.

28381 The <length> field in the extended header record was included to make it simpler to step  
 28382 through the records, even if a record contains an unknown format (to a particular *pax*) with  
 28383 complex interactions of special characters. It also provides a minor integrity checkpoint within  
 28384 the records to aid a program attempting to recover files from a damaged archive.

28385 There are no extended header versions of the *devmajor* and *devminor* fields because the  
 28386 unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific  
 28387 extended keywords (such as *VENDOR.devmajor*) should be used.

28388 Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly  
 28389 on a symbolic name basis, as in **ustar**.

28390 Just as with the **ustar** format descriptions, the new format makes no special arrangements for  
 28391 multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file  
 28392 and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing  
 28393 their labels, and mounting each in the proper sequence are considered to be implementation  
 28394 details that cannot be described portably.

28395 The **pax** format is intended for interchange, not only for backup on a single (family of) systems.  
 28396 It is not as densely packed as might be possible for backup:

- 28397 • It contains information as coded characters that could be coded in binary.
- 28398 • It identifies extended records with name fields that could be omitted in favor of a fixed-field  
 28399 layout.
- 28400 • It translates names into a portable character set and identifies locale-related information,  
 28401 both of which are probably unnecessary for backup.

28402 The requirements on restoring from an archive are slightly different from the historical wording,  
 28403 allowing for non-monolithic privilege to bring forward as much as possible. In particular,  
 28404 attributes such as “high performance file” might be broadly but not universally granted while

28405 set-user-ID or *chown()* might be much more restricted. There is no implication in  
28406 IEEE Std 1003.1-2001 that the security information be honored after it is restored to the file  
28407 hierarchy, in spite of what might be improperly inferred by the silence on that topic. That is a  
28408 topic for another standard.

28409 Links are recorded in the fashion described here because a link can be to any file type. It is  
28410 desirable in general to be able to restore part of an archive selectively and restore all of those  
28411 files completely. If the data is not associated with each link, it is not possible to do this.  
28412 However, the data associated with a file can be large, and when selective restoration is not  
28413 needed, this can be a significant burden. The archive is structured so that files that have no  
28414 associated data can always be restored by the name of any link name of any link, and the user  
28415 may choose whether data is recorded with each instance of a file that contains data. The format  
28416 permits mixing of both types of links in a single archive; this can be done for special needs, and  
28417 *pax* is expected to interpret such archives on input properly, despite the fact that there is no *pax*  
28418 option that would force this mixed case on output. (When **-o linkdata** is used, the output must  
28419 contain the duplicate data, but the implementation is free to include it or omit it when **-o**  
28420 **linkdata** is not used.)

28421 The time values are included as extended header records for those implementations needing  
28422 more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be  
28423 negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject  
28424 the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a  
28425 leading '-'. Even though some implementations can support finer file-time granularities than  
28426 seconds, the normative text requires support only for seconds since the Epoch because the  
28427 ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new  
28428 format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will  
28429 be affected by the **-p a** and **-p e** options. The *ctime* creation time (actually *inode* modification  
28430 time) is described with “appropriate privilege” so that it can be ignored when writing to the file  
28431 system. POSIX does not provide a portable means to change file creation time. Nothing is  
28432 intended to prevent a non-portable implementation of *pax* from restoring the value.

28433 The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the  
28434 sizes specified in the regular *tar* header. New file system architectures are emerging that will  
28435 exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits  
28436 for user and group IDs, but the extended header values were included for completeness,  
28437 allowing overrides for all of the decimal values in the *tar* header.

28438 The standard developers intended to describe the effective results of *pax* with regard to file  
28439 ownerships and permissions; implementations are not restricted in timing or sequencing the  
28440 restoration of such, provided the results are as specified.

28441 Much of the text describing the extended headers refers to use in “**write** or **copy** modes”. The  
28442 **copy** mode references are due to the normative text: “The effect of the copy shall be as if the  
28443 copied files were written to an archive file and then subsequently extracted ...”. There is  
28444 certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode,  
28445 but the effects must be as if it had.

**pax Archive Character Set Encoding/Decoding**

There is a need to exchange archives of files between systems of different native codesets. Filenames, group names, and user names must be preserved to the fullest extent possible when an archive is read on the receiving platform. Translation of the contents of files is not within the scope of the *pax* utility.

There will also be the need to represent characters that are not available on the receiving platform. These unsupported characters cannot be automatically folded to the local set of characters due to the chance of collisions. This could result in overwriting previous extracted files from the archive or pre-existing files on the system.

For these reasons, the codeset used to represent characters within the extended header records of the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields requiring translation include, at a minimum, filenames, user names, group names, and link pathnames. Implementations may wish to have localized extended keywords that use non-portable characters.

The standard developers considered the following options:

- The archive creator specifies the well-defined name of the source codeset. The receiver must then recognize the codeset name and perform the appropriate translations to the destination codeset.
- The archive creator includes within the archive the character mapping table for the source codeset used to encode extended header records. The receiver must then read the character mapping table and perform the appropriate translations to the destination codeset.
- The archive creator translates the extended header records in the source codeset into a canonical form. The receiver must then perform the appropriate translations to the destination codeset.

The approach that incorporates the name of the source codeset poses the problem of codeset name registration, and makes the archive useless to *pax* archive decoders that do not recognize that codeset.

Because parts of an archive may be corrupted, the standard developers felt that including the character map of the source codeset was too fragile. The loss of this one key component could result in making the entire archive useless. (The difference between this and the global extended header decision was that the latter has a workaround—duplicating extended header records on unreliable media—but this would be too burdensome for large character set maps.)

Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the cross-product of all source and destination codesets.

To simplify the translation from the source codeset to the canonical form and from the canonical form to the destination codeset, the standard developers decided that the internal representation should be a stateless encoding. A stateless encoding is one where each codepoint has the same meaning, without regard to the decoder being in a specific state. An example of a stateful encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the ISO/IEC 646:1991 standard (equivalent to 7-bit ASCII).

For these reasons, the standard developers decided to adopt a canonical format for the representation of file information strings. The obvious, well-endorsed candidate is the ISO/IEC 10646-1:2000 standard (based in part on Unicode), which can be used to represent the characters of virtually all standardized character sets. The standard developers initially agreed upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters provides a sufficiently rich set to represent all commonly-used codesets.



However, the standard developers found that the 16-bit Unicode representation had some problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character made the extended header records twice as long for the case of strings coded entirely from historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the ISO/IEC 10646-1:2000 standard. This multi-byte representation encodes UCS2 or UCS4 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In addition, NUL octets and other characters possibly confusing to POSIX file systems do not appear, except to represent themselves. It was realized that certain national codesets take up more space after the encoding, due to their placement within the UCS range; it was felt that the usefulness of the encoding of the names outweighs the disadvantage of size increase for file, user, and group names.

The encoding of UTF-8 is as follows:

| UCS4 Hex Encoding | UTF-8 Binary Encoding                                 |
|-------------------|-------------------------------------------------------|
| 00000000-0000007F | 0xxxxxxx                                              |
| 00000080-000007FF | 110xxxxx 10xxxxxx                                     |
| 00000800-0000FFFF | 1110xxxx 10xxxxxx 10xxxxxx                            |
| 00010000-001FFFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx                   |
| 00200000-03FFFFFF | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx          |
| 04000000-7FFFFFFF | 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

where each 'x' represents a bit value from the character being translated.

### ustar Interchange Format

The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of the historical *tar* utility. The goal of these changes was not only to provide the functional enhancements desired, but also to retain compatibility between new and old versions. This compatibility has been retained. Archives written using the old archive format are compatible with the new format.

Implementors should be aware that the previous file format did not include a mechanism to archive directory type files. For this reason, the convention of using a filename ending with slash was adopted to specify a directory on the archive.

The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for {PATH\_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname be stored there without the use of the *prefix* field. Although the name field is known to be too small to contain {PATH\_MAX} characters, the value was not changed in this version of the archive file format to retain backwards-compatibility, and instead the prefix was introduced. Also, because of the earlier version of the format, there is no way to remove the restriction on the *linkname* field being limited in size to just that of the *name* field.

The *size* field is required to be meaningful in all implementation extensions, although it could be zero. This is required so that the data blocks can always be properly counted.

It is suggested that if device special files need to be represented that cannot be represented in the standard format, that one of the extension types (**A-Z**) be used, and that the additional information for the special file be represented as data and be reflected in the *size* field.

Attempting to restore a special file type, where it is converted to ordinary data and conflicts with an existing filename, need not be specially detected by the utility. If run as an ordinary user, *pax* should not be able to overwrite the entries in, for example, */dev* in any case (whether the file is converted to another type or not). If run as a privileged user, it should be able to do so, and it would be considered a bug if it did not. The same is true of ordinary data files and similarly

named special files; it is impossible to anticipate the needs of the user (who could really intend to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the protection system as required.

The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a **ustar** archive. IEEE Std 1003.1-2001 does not require the contiguous file extension, but does define a standard way of archiving such files so that all conforming systems can interpret these file types in a meaningful and consistent manner. On a system that does not support extended file types, the *pax* utility should do the best it can with the file and go on to the next.

The file protection modes are those conventionally used by the *ls* utility. This is extended beyond the usage in the ISO POSIX-2 standard to support the “shared text” or “sticky” bit. It is intended that the conformance document should not document anything beyond the existence of and support of such a mode. Further extensions are expected to these bits, particularly with overloading the set-user-ID and set-group-ID flags.

### 28551 **cpio Interchange Format**

The reference to appropriate privilege in the **cpio** format refers to an error on standard output; the **ustar** format does not make comparable statements.

The model for this format was the historical System V *cpio-c* data interchange format. This model documents the portable version of the **cpio** format and not the binary version. It has the flexibility to transfer data of any type described within IEEE Std 1003.1-2001, yet is extensible to transfer data types specific to extensions beyond IEEE Std 1003.1-2001 (for example, contiguous files). Because it describes existing practice, there is no question of maintaining upwards-compatibility.

### 28560 **cpio Header**

There has been some concern that the size of the *c\_ino* field of the header is too small to handle those systems that have very large *inode* numbers. However, the *c\_ino* field in the header is used strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as the *inode* number of the file in the location from which that file is extracted.

The name *c\_magic* is based on historical usage.

### 28566 **cpio Filename**

For most historical implementations of the *cpio* utility, {PATH\_MAX} octets can be used to describe the pathname without the addition of any other header fields (the NUL character would be included in this count). {PATH\_MAX} is the minimum value for pathname size, documented as 256 bytes. However, an implementation may use *c\_namesize* to determine the exact length of the pathname. With the current description of the **<cpio.h>** header, this pathname size can be as large as a number that is described in six octal digits.

Two values are documented under the *c\_mode* field values to provide for extensibility for known file types:

**0110 000**      Reserved for contiguous files. The implementation may treat the rest of the information for this archive like a regular file. If this file type is undefined, the implementation may create the file as a regular file.

This provides for extensibility of the **cpio** format while allowing for the ability to read old archives. Files of an unknown type may be read as “regular files” on some implementations. On a system that does not support extended file types, the *pax* utility should do the best it can with the file and go on to the next.

28582 **FUTURE DIRECTIONS**

28583 None.

28584 **SEE ALSO**

28585 Chapter 2 (on page 29), *cp*, *ed*, *getopts*, *ls*, *printf*, the Base Definitions volume of  
 28586 IEEE Std 1003.1-2001, <**cpio.h**>, the System Interfaces volume of IEEE Std 1003.1-2001, *chown()*,  
 28587 *creat()*, *mkdir()*, *mkfifo()*, *stat()*, *utime()*, *write()*

28588 **CHANGE HISTORY**

28589 First released in Issue 4.

28590 **Issue 5**

28591 A note is added to the APPLICATION USAGE indicating that the **cpio** and **tar** formats can only  
 28592 support files up to 8 gigabytes in size.

28593 **Issue 6**28594 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- 28595 • Support has been added for symbolic links in the options and interchange formats.
- 28596 • A new format has been devised, based on extensions to **ustar**.
- 28597 • References to the “extended” **tar** and **cpio** formats derived from the POSIX.1-1990 standard  
 28598 have been changed to remove the “extended” adjective because this could cause confusion  
 28599 with the extended *tar* header added in this revision. (All references to *tar* are actually to  
 28600 **ustar**.)

28601 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

28602 IEEE PASC Interpretation 1003.2 #168 is applied, clarifying that *mkdir()* and *mkfifo()* calls can  
 28603 ignore an [EEXIST] error when extracting an archive.

28604 IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when  
 28605 in **read** mode.

28606 IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **-t** option.

28607 IEEE PASC Interpretation 1003.2 #195 is applied.

28608 IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **-H**, **-L**,  
 28609 and **-I** options.

28610 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/35 is applied, adding the process ID of  
 28611 the *pax* process into certain fields. This change provides a method for the implementation to  
 28612 ensure that different instances of *pax* extracting a file named **/a/b/foo** will not collide when  
 28613 processing the extended header information associated with **foo**.

28614 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/36 is applied, changing **-x B** to **-x pax** in  
 28615 the OPTIONS section.

28616 **NAME**

28617 pr — print files

28618 **SYNOPSIS**

28619 pr [+page] [-column] [-adFmrt] [-e[*char*] [*gap*]] [-h *header*] [-i[*char*] [*gap*]]  
 28620 xSI [-l *lines*] [-n[*char*] [*width*]] [-o *offset*] [-s[*char*]] [-w *width*] [-fp]  
 28621 [*file...*]

28622 **DESCRIPTION**

28623 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be  
 28624 read, formatted, and written to standard output. By default, the input shall be separated into 66-  
 28625 line pages, each with:

- 28626 • A 5-line header that includes the page number, date, time, and the pathname of the file
- 28627 • A 5-line trailer consisting of blank lines

28628 If standard output is associated with a terminal, diagnostic messages shall be deferred until the  
 28629 *pr* utility has completed processing.

28630 When options specifying multi-column output are specified, output text columns shall be of  
 28631 equal width; input lines that do not fit into a text column shall be truncated. By default, text  
 28632 columns shall be separated with at least one <blank>.

28633 **OPTIONS**

28634 The *pr* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 28635 Utility Syntax Guidelines, except that: the *page* option has a '+' delimiter; *page* and *column* can  
 28636 be multi-digit numbers; some of the option-arguments are optional; and some of the option-  
 28637 arguments cannot be specified as separate arguments from the preceding option letter. In  
 28638 particular, the *-s* option does not allow the option letter to be separated from its argument, and  
 28639 the options *-e*, *-i*, and *-n* require that both arguments, if present, not be separated from the  
 28640 option letter.

28641 The following options shall be supported. In the following option descriptions, *column*, *lines*,  
 28642 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

28643 *+page* Begin output at page number *page* of the formatted input.

28644 *-column* Produce multi-column output that is arranged in *column* columns (the default shall  
 28645 be 1) and is written down each column in the order in which the text is received  
 28646 from the input file. This option should not be used with *-m*. The options *-e* and *-i*  
 28647 shall be assumed for multiple text-column output. Whether or not text columns  
 28648 are produced with identical vertical lengths is unspecified, but a text column shall  
 28649 never exceed the length of the page (see the *-l* option). When used with *-t*, use the  
 28650 minimum number of lines to write the output.

28651 *-a* Modify the effect of the *-column* option so that the columns are filled across the  
 28652 page in a round-robin order (for example, when *column* is 2, the first input line  
 28653 heads column 1, the second heads column 2, the third is the second line in column  
 28654 1, and so on).

28655 *-d* Produce output that is double-spaced; append an extra <newline> following every  
 28656 <newline> found in the input.

28657 *-e[*char*][*gap*]*

28658 Expand each input <tab> to the next greater column position specified by the  
 28659 formula  $n*gap+1$ , where *n* is an integer > 0. If *gap* is zero or is omitted, it shall  
 28660 default to 8. All <tab>s in the input shall be expanded into the appropriate number  
 28661 of <space>s. If any non-digit character, *char*, is specified, it shall be used as the

|           |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 28662     |                                      | input <tab>.                                                                                                                                                                                                                                                                                                                                                                                    |
| 28663 XSI | <b>-f</b>                            | Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline>s. Pause before beginning the first page if the standard output is associated with a terminal.                                                                                                                                                                                                |
| 28664     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28665     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28666     | <b>-F</b>                            | Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline>s.                                                                                                                                                                                                                                                                                            |
| 28667     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28668     | <b>-h header</b>                     | Use the string <i>header</i> to replace the contents of the <i>file</i> operand in the page header.                                                                                                                                                                                                                                                                                             |
| 28669     | <b>-i[<i>char</i>][<i>gap</i>]</b>   | In output, replace multiple <space>s with <tab>s wherever two or more adjacent <space>s reach column positions <i>gap</i> +1, 2* <i>gap</i> +1, 3* <i>gap</i> +1, and so on. If <i>gap</i> is zero or is omitted, default tab settings at every eighth column position shall be assumed. If any non-digit character, <i>char</i> , is specified, it shall be used as the output <tab>.          |
| 28670     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28671     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28672     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28673     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28674     | <b>-l lines</b>                      | Override the 66-line default and reset the page length to <i>lines</i> . If <i>lines</i> is not greater than the sum of both the header and trailer depths (in lines), the <i>pr</i> utility shall suppress both the header and trailer, as if the <b>-t</b> option were in effect.                                                                                                             |
| 28675     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28676     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28677     | <b>-m</b>                            | Merge files. Standard output shall be formatted so the <i>pr</i> utility writes one line from each file specified by a <i>file</i> operand, side by side into text columns of equal fixed widths, in terms of the number of column positions. Implementations shall support merging of at least nine <i>file</i> operands.                                                                      |
| 28678     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28679     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28680     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28681     | <b>-n[<i>char</i>][<i>width</i>]</b> |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28682     |                                      | Provide <i>width</i> -digit line numbering (default for <i>width</i> shall be 5). The number shall occupy the first <i>width</i> column positions of each text column of default output or each line of <b>-m</b> output. If <i>char</i> (any non-digit character) is given, it shall be appended to the line number to separate it from whatever follows (default for <i>char</i> is a <tab>). |
| 28683     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28684     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28685     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28686     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28687     | <b>-o offset</b>                     | Each line of output shall be preceded by offset <space>s. If the <b>-o</b> option is not specified, the default offset shall be zero. The space taken is in addition to the output line width (see the <b>-w</b> option below).                                                                                                                                                                 |
| 28688     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28689     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28690     | <b>-p</b>                            | Pause before beginning each page if the standard output is directed to a terminal ( <i>pr</i> shall write an <alert> to standard error and wait for a <carriage-return> to be read on <b>/dev/tty</b> ).                                                                                                                                                                                        |
| 28691     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28692     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28693     | <b>-r</b>                            | Write no diagnostic reports on failure to open files.                                                                                                                                                                                                                                                                                                                                           |
| 28694     | <b>-s[<i>char</i>]</b>               | Separate text columns by the single character <i>char</i> instead of by the appropriate number of <space>s (default for <i>char</i> shall be <tab>).                                                                                                                                                                                                                                            |
| 28695     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28696     | <b>-t</b>                            | Write neither the five-line identifying header nor the five-line trailer usually supplied for each page. Quit writing after the last line of each file without spacing to the end of the page.                                                                                                                                                                                                  |
| 28697     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28698     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28699     | <b>-w width</b>                      | Set the width of the line to <i>width</i> column positions for multiple text-column output only. If the <b>-w</b> option is not specified and the <b>-s</b> option is not specified, the default width shall be 72. If the <b>-w</b> option is not specified and the <b>-s</b> option is specified, the default width shall be 512.                                                             |
| 28700     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28701     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28702     |                                      |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28703     |                                      | For single column output, input lines shall not be truncated.                                                                                                                                                                                                                                                                                                                                   |

28704 **OPERANDS**

28705 The following operand shall be supported:

28706 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*  
 28707 operand is '-', the standard input shall be used.

28708 **STDIN**

28709 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
 28710 See the INPUT FILES section.

28711 **INPUT FILES**

28712 The input files shall be text files.

28713 The file **/dev/tty** shall be used to read responses required by the **-p** option.

28714 **ENVIRONMENT VARIABLES**

28715 The following environment variables shall affect the execution of *pr*:

28716 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 28717 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 28718 Internationalization Variables for the precedence of internationalization variables  
 28719 used to determine the values of locale categories.)

28720 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 28721 internationalization variables.

28722 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 28723 characters (for example, single-byte as opposed to multi-byte characters in  
 28724 arguments and input files) and which characters are defined as printable (character  
 28725 class **print**). Non-printable characters are still written to standard output, but are  
 28726 not counted for the purpose for column-width and line-length calculations.

28727 *LC\_MESSAGES*  
 28728 Determine the locale that should be used to affect the format and contents of  
 28729 diagnostic messages written to standard error.

28730 *LC\_TIME* Determine the format of the date and time for use in writing header lines.

28731 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

28732 *TZ* Determine the timezone used to calculate date and time strings written in header  
 28733 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.

28734 **ASYNCHRONOUS EVENTS**

28735 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error  
 28736 messages to the screen before terminating.

28737 **STDOUT**

28738 The *pr* utility output shall be a paginated version of the original file (or files). This pagination  
 28739 shall be accomplished using either <form-feed>s or a sequence of <newline>s, as controlled by  
 28740 XSI the **-F** or **-f** option. Page headers shall be generated unless the **-t** option is specified. The page  
 28741 headers shall be of the form:

28742 "\n\n%s %s Page %d\n\n\n", <output of date>, <file>, <page number>

28743 In the POSIX locale, the <output of date> field, representing the date and time of last modification  
 28744 of the input file (or the current date and time if the input file is standard input), shall be  
 28745 equivalent to the output of the following command as it would appear if executed at the given  
 28746 time:

28747           date "+%b %e %H:%M %Y"

28748           without the trailing <newline>, if the page being written is from standard input. If the page  
28749           being written is not from standard input, in the POSIX locale, the same format shall be used, but  
28750           the time used shall be the modification time of the file corresponding to *file* instead of the current  
28751           time. When the *LC\_TIME* locale category is not set to the POSIX locale, a different format and  
28752           order of presentation of this field may be used.

28753           If the standard input is used instead of a *file* operand, the <*file*> field shall be replaced by a null  
28754           string.

28755           If the **-h** option is specified, the <*file*> field shall be replaced by the *header* argument.

#### 28756 **STDERR**

28757           The standard error shall be used for diagnostic messages and for alerting the terminal when **-p**  
28758           is specified.

#### 28759 **OUTPUT FILES**

28760           None.

#### 28761 **EXTENDED DESCRIPTION**

28762           None.

#### 28763 **EXIT STATUS**

28764           The following exit values shall be returned:

28765           0   Successful completion.

28766           >0   An error occurred.

#### 28767 **CONSEQUENCES OF ERRORS**

28768           Default.

#### 28769 **APPLICATION USAGE**

28770           None.

#### 28771 **EXAMPLES**

28772           1.   Print a numbered list of all files in the current directory:

28773                ls -a | pr -n -h "Files in \$(pwd) ."

28774           2.   Print **file1** and **file2** as a double-spaced, three-column listing headed by “file list”:

28775                pr -3d -h "file list" file1 file2

28776           3.   Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ...:

28777                pr -e9 -t <file1 >file2

#### 28778 **RATIONALE**

28779           This utility is one of those that does not follow the Utility Syntax Guidelines because of its  
28780           historical origins. The standard developers could have added new options that obeyed the  
28781           guidelines (and marked the old options obsolescent) or devised an entirely new utility; there are  
28782           examples of both actions in this volume of IEEE Std 1003.1-2001. Because of its widespread use  
28783           by historical applications, the standard developers decided to exempt this version of *pr* from  
28784           many of the guidelines.

28785           Implementations are required to accept option-arguments to the **-h**, **-l**, **-o**, and **-w** options  
28786           whether presented as part of the same argument or as a separate argument to *pr*, as suggested by  
28787           the Utility Syntax Guidelines. The **-n** and **-s** options, however, are specified as in historical  
28788           practice because they are frequently specified without their optional arguments. If a <blank>

28789 were allowed before the option-argument in these cases, a *file* operand could mistakenly be  
28790 interpreted as an option-argument in historical applications.

28791 The text about the minimum number of lines in multi-column output was included to ensure  
28792 that a best effort is made in balancing the length of the columns. There are known historical  
28793 implementations in which, for example, 60-line files are listed by *pr -2* as one column of 56 lines  
28794 and a second of 4. Although this is not a problem when a full page with headers and trailers is  
28795 produced, it would be relatively useless when used with *-t*.

28796 Historical implementations of the *pr* utility have differed in the action taken for the *-f* option.  
28797 BSD uses it as described here for the *-F* option; System V uses it to change trailing <newline>s  
28798 on each page to a <form-feed> and, if standard output is a TTY device, sends an <alert> to  
28799 standard error and reads a line from */dev/tty* before the first page. There were strong arguments  
28800 from both sides of this issue concerning historical practice and as a result the *-F* option was  
28801 added. XSI-conformant systems support the System V historical actions for the *-f* option.

28802 The <output of date> field in the *-I* format is specified only for the POSIX locale. As noted, the  
28803 format can be different in other locales. No mechanism for defining this is present in this volume  
28804 of IEEE Std 1003.1-2001, as the appropriate vehicle is a message catalog; that is, the format  
28805 should be specified as a “message”.

#### 28806 **FUTURE DIRECTIONS**

28807 None.

#### 28808 **SEE ALSO**

28809 *expand*, *lp*

#### 28810 **CHANGE HISTORY**

28811 First released in Issue 2.

#### 28812 **Issue 6**

28813 The following new requirements on POSIX implementations derive from alignment with the  
28814 Single UNIX Specification:

- 28815 • The *-p* option is added.

28816 The normative text is reworded to avoid use of the term “must” for application requirements.



28817 **NAME**

28818           printf — write formatted output

28819 **SYNOPSIS**28820           printf *format* [*argument...*]28821 **DESCRIPTION**

28822           The *printf* utility shall write formatted operands to the standard output. The *argument* operands  
 28823           shall be formatted under control of the *format* operand.

28824 **OPTIONS**

28825           None.

28826 **OPERANDS**

28827           The following operands shall be supported:

28828           *format*           A string describing the format to use to write the remaining operands. See the  
 28829           EXTENDED DESCRIPTION section.

28830           *argument*       The strings to be written to standard output, under the control of *format*. See the  
 28831           EXTENDED DESCRIPTION section.

28832 **STDIN**

28833           Not used.

28834 **INPUT FILES**

28835           None.

28836 **ENVIRONMENT VARIABLES**28837           The following environment variables shall affect the execution of *printf*:

28838           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 28839           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 28840           Internationalization Variables for the precedence of internationalization variables  
 28841           used to determine the values of locale categories.)

28842           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 28843           internationalization variables.

28844           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 28845           characters (for example, single-byte as opposed to multi-byte characters in  
 28846           arguments).

28847           *LC\_MESSAGES*

28848                           Determine the locale that should be used to affect the format and contents of  
 28849           diagnostic messages written to standard error.

28850           *LC\_NUMERIC*

28851                           Determine the locale for numeric formatting. It shall affect the format of numbers  
 28852           written using the e, E, f, g, and G conversion specifier characters (if supported).

28853 *XSI*           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.28854 **ASYNCHRONOUS EVENTS**

28855           Default.

28856 **STDOUT**

28857           See the EXTENDED DESCRIPTION section.

28858 **STDERR**

28859 The standard error shall be used only for diagnostic messages.

28860 **OUTPUT FILES**

28861 None.

28862 **EXTENDED DESCRIPTION**

28863 The *format* operand shall be used as the *format* string described in the Base Definitions volume of  
 28864 IEEE Std 1003.1-2001, Chapter 5, File Format Notation with the following exceptions:

- 28865 1. A <space> in the format string, in any context other than a flag of a conversion  
 28866 specification, shall be treated as an ordinary character that is copied to the output.
- 28867 2. A 'Δ' character in the format string shall be treated as a 'Δ' character, not as a <space>.
- 28868 3. In addition to the escape sequences shown in the Base Definitions volume of  
 28869 IEEE Std 1003.1-2001, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n',  
 28870 '\r', '\t', '\v'), "\ddd", where *ddd* is a one, two, or three-digit octal number, shall be  
 28871 written as a byte with the numeric value specified by the octal number.
- 28872 4. The implementation shall not precede or follow output from the *d* or *u* conversion  
 28873 specifiers with <blank>s not specified by the *format* operand.
- 28874 5. The implementation shall not precede output from the *o* conversion specifier with zeros  
 28875 not specified by the *format* operand.
- 28876 6. The *e*, *E*, *f*, *g*, and *G* conversion specifiers need not be supported.
- 28877 7. An additional conversion specifier character, *b*, shall be supported as follows. The  
 28878 argument shall be taken to be a string that may contain backslash-escape sequences. The  
 28879 following backslash-escape sequences shall be supported:
  - 28880 — The escape sequences listed in the Base Definitions volume of IEEE Std 1003.1-2001,  
 28881 Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'),  
 28882 which shall be converted to the characters they represent
  - 28883 — "\0ddd", where *ddd* is a zero, one, two, or three-digit octal number that shall be  
 28884 converted to a byte with the numeric value specified by the octal number
  - 28885 — '\c', which shall not be written and shall cause *printf* to ignore any remaining  
 28886 characters in the string operand containing it, any remaining string operands, and any  
 28887 additional characters in the *format* operand
- 28888 The interpretation of a backslash followed by any other sequence of characters is  
 28889 unspecified.
- 28890 Bytes from the converted string shall be written until the end of the string or the number of  
 28891 bytes indicated by the precision specification is reached. If the precision is omitted, it shall  
 28892 be taken to be infinite, so all bytes up to the end of the converted string shall be written.
- 28893 8. For each conversion specification that consumes an argument, the next argument operand  
 28894 shall be evaluated and converted to the appropriate type for the conversion as specified  
 28895 below.
- 28896 9. The *format* operand shall be reused as often as necessary to satisfy the argument operands.  
 28897 Any extra *c* or *s* conversion specifiers shall be evaluated as if a null string argument were  
 28898 supplied; other extra conversion specifications shall be evaluated as if a zero argument  
 28899 were supplied. If the *format* operand contains no conversion specifications and *argument*  
 28900 operands are present, the results are unspecified.

28901 10. If a character sequence in the *format* operand begins with a '%' character, but does not  
28902 form a valid conversion specification, the behavior is unspecified.

28903 The *argument* operands shall be treated as strings if the corresponding conversion specifier is b,  
28904 c, or s; otherwise, it shall be evaluated as a C constant, as described by the ISO C standard, with  
28905 the following extensions:

- 28906 • A leading plus or minus sign shall be allowed.
- 28907 • If the leading character is a single-quote or double-quote, the value shall be the numeric  
28908 value in the underlying codeset of the character following the single-quote or double-quote.

28909 If an argument operand cannot be completely converted into an internal value appropriate to  
28910 the corresponding conversion specification, a diagnostic message shall be written to standard  
28911 error and the utility shall not exit with a zero exit status, but shall continue processing any  
28912 remaining operands and shall write the value accumulated at the time the error was detected to  
28913 standard output.

28914 It is not considered an error if an argument operand is not completely used for a c or s  
28915 conversion or if a string operand's first or second character is used to get the numeric value of a  
28916 character.

#### 28917 EXIT STATUS

28918 The following exit values shall be returned:

- 28919 0 Successful completion.
- 28920 >0 An error occurred.

#### 28921 CONSEQUENCES OF ERRORS

28922 Default.

#### 28923 APPLICATION USAGE

28924 The floating-point formatting conversion specifications of *printf()* are not required because all  
28925 arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations  
28926 and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-  
28927 point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility  
28928 cannot really be used to format *bc* output; it does not support arbitrary precision.)  
28929 Implementations are encouraged to support the floating-point conversions as an extension.

28930 Note that this *printf* utility, like the *printf()* function defined in the System Interfaces volume of  
28931 IEEE Std 1003.1-2001 on which it is based, makes no special provision for dealing with multi-  
28932 byte characters when using the %c conversion specification or when a precision is specified in a  
28933 %b or %s conversion specification. Applications should be extremely cautious using either of  
28934 these features when there are multi-byte characters in the character set.

28935 No provision is made in this volume of IEEE Std 1003.1-2001 which allows field widths and  
28936 precisions to be specified as '\*' since the '\*' can be replaced directly in the *format* operand  
28937 using shell variable substitution. Implementations can also provide this feature as an extension  
28938 if they so choose.

28939 Hexadecimal character constants as defined in the ISO C standard are not recognized in the  
28940 *format* operand because there is no consistent way to detect the end of the constant. Octal  
28941 character constants are limited to, at most, three octal digits, but hexadecimal character  
28942 constants are only terminated by a non-hex-digit character. In the ISO C standard, the "##"  
28943 concatenation operator can be used to terminate a constant and follow it with a hexadecimal  
28944 character to be written. In the shell, concatenation occurs before the *printf* utility has a chance to  
28945 parse the end of the hexadecimal constant.

The %b conversion specification is not part of the ISO C standard; it has been added here as a portable way to process backslash escapes expanded in string operands as provided by the *echo* utility. See also the APPLICATION USAGE section of *echo* (on page 333) for ways to use *printf* as a replacement for all of the traditional versions of the *echo* utility.

If an argument cannot be parsed correctly for the corresponding conversion specification, the *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end of an argument being used for a numeric conversion shall be reported as errors.

#### EXAMPLES

To alert the user and then print and read a series of prompts:

```
printf "\aPlease fill in the following: \nName: "
read name
printf "Phone number: "
read phone
```

To read out a list of right and wrong answers from a file, calculate the percentage correctly, and print them out. The numbers are right-justified and separated by a single <tab>. The percentage is written to one decimal place of accuracy:

```
while read right wrong ; do
 percent=$(echo "scale=1; ($right*100)/($right+$wrong)" | bc)
 printf "%2d right\t%2d wrong\t(%%s%%)\n" \
 $right $wrong $percent
done < database_file
```

The command:

```
printf "%5d%4d\n" 1 21 321 4321 54321
```

produces:

```
 1 21
 3214321
54321 0
```

Note that the *format* operand is used three times to print all of the given strings and that a '0' was supplied by *printf* to satisfy the last %4d conversion specification.

The *printf* utility is required to notify the user when conversion errors are detected while producing numeric output; thus, the following results would be expected on an implementation with 32-bit twos-complement integers when %d is specified as the *format* operand:

| Argument    | Standard Output | Diagnostic Output                         |
|-------------|-----------------|-------------------------------------------|
| 5a          | 5               | printf: "5a" not completely converted     |
| 9999999999  | 2147483647      | printf: "9999999999" arithmetic overflow  |
| -9999999999 | -2147483648     | printf: "-9999999999" arithmetic overflow |
| ABC         | 0               | printf: "ABC" expected numeric value      |

The diagnostic message format is not specified, but these examples convey the type of information that should be reported. Note that the value shown on standard output is what would be expected as the return value from the *strtol()* function as defined in the System Interfaces volume of IEEE Std 1003.1-2001. A similar correspondence exists between %u and *strtoul()* and %e, %f, and %g (if the implementation supports floating-point conversions) and *strtod()*.

28990 In a locale using the ISO / IEC 646: 1991 standard as the underlying codeset, the command:

28991 `printf "%d\n" 3 +3 -3 \'3 \"+3 \"'-3"`

28992 produces:

28993 3 Numeric value of constant 3

28994 3 Numeric value of constant 3

28995 -3 Numeric value of constant -3

28996 51 Numeric value of the character '3' in the ISO / IEC 646: 1991 standard codeset

28997 43 Numeric value of the character '+' in the ISO / IEC 646: 1991 standard codeset

28998 45 Numeric value of the character '-' in the ISO / IEC 646: 1991 standard codeset

28999 Note that in a locale with multi-byte characters, the value of a character is intended to be the  
29000 value of the equivalent of the **wchar\_t** representation of the character as described in the System  
29001 Interfaces volume of IEEE Std 1003.1-2001.

#### 29002 **RATIONALE**

29003 The *printf* utility was added to provide functionality that has historically been provided by *echo*.  
29004 However, due to irreconcilable differences in the various versions of *echo* extant, the version has  
29005 few special features, leaving those to this new *printf* utility, which is based on one in the Ninth  
29006 Edition system.

29007 The EXTENDED DESCRIPTION section almost exactly matches the *printf()* function in the  
29008 ISO C standard, although it is described in terms of the file format notation in the Base  
29009 Definitions volume of IEEE Std 1003.1-2001, Chapter 5, File Format Notation.

#### 29010 **FUTURE DIRECTIONS**

29011 None.

#### 29012 **SEE ALSO**

29013 *awk*, *bc*, *echo*, the System Interfaces volume of IEEE Std 1003.1-2001, *printf()*

#### 29014 **CHANGE HISTORY**

29015 First released in Issue 4.

29016 **NAME**29017        *prs* — print an SCCS file (**DEVELOPMENT**)29018 **SYNOPSIS**29019 xSI        *prs* [-a] [-d *dataspec*] [-r[*SID*]] *file...*29020 xSI        *prs* [ -e | -l ] -c *cutoff* [-d *dataspec*] *file...*29021 xSI        *prs* [ -e | -l ] -r[*SID*] [-d *dataspec*] *file...*29022 **DESCRIPTION**29023        The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied  
29024        format.29025 **OPTIONS**29026        The *prs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
29027        Utility Syntax Guidelines, except that the **-r** option has an optional option-argument. This  
29028        optional option-argument cannot be presented as a separate argument. The following options  
29029        shall be supported:29030        **-d** *dataspec*    Specify the output data specification. The *dataspec* shall be a string consisting of  
29031        SCCS file *data keywords* (see **Data Keywords** (on page 747)) interspersed with  
29032        optional user-supplied text.29033        **-r**[*SID*]        Specify the SCCS identification string (SID) of a delta for which information is  
29034        desired. If no *SID* option-argument is specified, the SID of the most recently  
29035        created delta shall be assumed.29036        **-e**                Request information for all deltas created earlier than and including the delta  
29037        designated via the **-r** option or the date-time given by the **-c** option.29038        **-l**                Request information for all deltas created later than and including the delta  
29039        designated via the **-r** option or the date-time given by the **-c** option.29040        **-c** *cutoff*        Indicate the *cutoff* date-time, in the form:29041        *YY*[*MM*][*DD*][*HH*][*MM*][*SS*]]]]29042        For the *YY* component, values in the range [69,99] shall refer to years 1969 to 1999  
29043        inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.29044        **Note:**        It is expected that in a future version of IEEE Std 1003.1-2001 the default  
29045        century inferred from a 2-digit year will change. (This would apply to all  
29046        commands accepting a 2-digit year as input.)29047        No changes (deltas) to the SCCS file that were created after the specified *cutoff*  
29048        date-time shall be included in the output. Units omitted from the date-time default  
29049        to their maximum possible values; for example, **-c 7502** is equivalent to  
29050        **-c 750228235959**.29051        **-a**                Request writing of information for both removed—that is, *delta type=R* (see  
29052        *rmDEL*)—and existing—that is, *delta type=D*,—deltas. If the **-a** option is not  
29053        specified, information for existing deltas only shall be provided.29054 **OPERANDS**

29055        The following operand shall be supported:

29056        *file*            A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs*  
29057        utility shall behave as though each file in the directory were specified as a named  
29058        file, except that non-SCCS files (last component of the pathname does not begin  
29059        with **s.**) and unreadable files shall be silently ignored.

29060 If exactly one *file* operand appears, and it is '*-*', the standard input shall be read;  
 29061 each line of the standard input shall be taken to be the name of an SCCS file to be  
 29062 processed. Non-SCCS files and unreadable files shall be silently ignored.

#### 29063 **STDIN**

29064 The standard input shall be a text file used only when the *file* operand is specified as '*-*'. Each  
 29065 line of the text file shall be interpreted as an SCCS pathname.

#### 29066 **INPUT FILES**

29067 Any SCCS files displayed are files of an unspecified format.

#### 29068 **ENVIRONMENT VARIABLES**

29069 The following environment variables shall affect the execution of *prs*:

29070 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 29071 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 29072 Internationalization Variables for the precedence of internationalization variables  
 29073 used to determine the values of locale categories.)

29074 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 29075 internationalization variables.

29076 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 29077 characters (for example, single-byte as opposed to multi-byte characters in  
 29078 arguments and input files).

29079 *LC\_MESSAGES*

29080 Determine the locale that should be used to affect the format and contents of  
 29081 diagnostic messages written to standard error.

29082 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 29083 **ASYNCHRONOUS EVENTS**

29084 Default.

#### 29085 **STDOUT**

29086 The standard output shall be a text file whose format is dependent on the data keywords  
 29087 specified with the *-d* option.

#### 29088 **Data Keywords**

29089 Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an  
 29090 SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple  
 29091 times.

29092 The information written by *prs* shall consist of:

- 29093 1. The user-supplied text
- 29094 2. Appropriate values (extracted from the SCCS file) substituted for the recognized data  
 29095 keywords in the order of appearance in the *dataspec*

29096 The format of a data keyword value shall either be simple ('*S*'), in which keyword substitution  
 29097 is direct, or multi-line ('*M*').

29098 User-supplied text shall be any text other than recognized data keywords. A <tab> shall be  
 29099 specified by '\t' and <newline> by '\n'. When the *-r* option is not specified, the default  
 29100 *dataspec* shall be:

29101 :PN: :\n\n

29102 and the following *dataspec* shall be used for each selected delta:

29103 :Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:

29104

29105

29106

29107

29108

29109

29110

29111

29112

29113

29114

29115

29116

29117

29118

29119

29120

29121

29122

29123

29124

29125

29126

29127

29128

29129

29130

29131

29132

29133

29134

29135

29136

29137

29138

29139

29140

29141

29142

29143

29144

29145

29146

29147

29148

| SCCS File Data Keywords |                                                          |              |                      |        |
|-------------------------|----------------------------------------------------------|--------------|----------------------|--------|
| Keyword                 | Data Item                                                | File Section | Value                | Format |
| <b>:Dt:</b>             | Delta information                                        | Delta Table  | See below*           | S      |
| <b>:DL:</b>             | Delta line statistics                                    | "            | <b>:Li:/Ld:/Lu:</b>  | S      |
| <b>:Li:</b>             | Lines inserted by Delta                                  | "            | <i>nnnnn***</i>      | S      |
| <b>:Ld:</b>             | Lines deleted by Delta                                   | "            | <i>nnnnn***</i>      | S      |
| <b>:Lu:</b>             | Lines unchanged by Delta                                 | "            | <i>nnnnn***</i>      | S      |
| <b>:DT:</b>             | Delta type                                               | "            | <b>D or R</b>        | S      |
| <b>:I:</b>              | SCCS ID string (SID)                                     | "            | See below**          | S      |
| <b>:R:</b>              | Release number                                           | "            | <i>nnnn</i>          | S      |
| <b>:L:</b>              | Level number                                             | "            | <i>nnnn</i>          | S      |
| <b>:B:</b>              | Branch number                                            | "            | <i>nnnn</i>          | S      |
| <b>:S:</b>              | Sequence number                                          | "            | <i>nnnn</i>          | S      |
| <b>:D:</b>              | Date delta created                                       | "            | <b>:Dy:/Dm:/Dd:</b>  | S      |
| <b>:Dy:</b>             | Year delta created                                       | "            | <i>nn</i>            | S      |
| <b>:Dm:</b>             | Month delta created                                      | "            | <i>nn</i>            | S      |
| <b>:Dd:</b>             | Day delta created                                        | "            | <i>nn</i>            | S      |
| <b>:T:</b>              | Time delta created                                       | "            | <b>:Th::Tm::Ts:</b>  | S      |
| <b>:Th:</b>             | Hour delta created                                       | "            | <i>nn</i>            | S      |
| <b>:Tm:</b>             | Minutes delta created                                    | "            | <i>nn</i>            | S      |
| <b>:Ts:</b>             | Seconds delta created                                    | "            | <i>nn</i>            | S      |
| <b>:P:</b>              | Programmer who created Delta                             | "            | <i>logname</i>       | S      |
| <b>:DS:</b>             | Delta sequence number                                    | "            | <i>nnnn</i>          | S      |
| <b>:DP:</b>             | Predecessor Delta sequence number                        | "            | <i>nnnn</i>          | S      |
| <b>:DI:</b>             | Sequence number of deltas included, excluded, or ignored | "            | <b>:Dn:/Dx:/Dg:</b>  | S      |
| <b>:Dn:</b>             | Deltas included (sequence #)                             | "            | <b>:DS: :DS: ...</b> | S      |
| <b>:Dx:</b>             | Deltas excluded (sequence #)                             | "            | <b>:DS: :DS: ...</b> | S      |
| <b>:Dg:</b>             | Deltas ignored (sequence #)                              | "            | <b>:DS: :DS: ...</b> | S      |
| <b>:MR:</b>             | MR numbers for delta                                     | "            | <i>text</i>          | M      |
| <b>:C:</b>              | Comments for delta                                       | "            | <i>text</i>          | M      |
| <b>:UN:</b>             | User names                                               | User Names   | <i>text</i>          | M      |
| <b>:FL:</b>             | Flag list                                                | Flags        | <i>text</i>          | M      |
| <b>:Y:</b>              | Module type flag                                         | "            | <i>text</i>          | S      |
| <b>:MF:</b>             | MR validation flag                                       | "            | <b>yes or no</b>     | S      |
| <b>:MP:</b>             | MR validation program name                               | "            | <i>text</i>          | S      |
| <b>:KF:</b>             | Keyword error, warning flag                              | "            | <b>yes or no</b>     | S      |
| <b>:KV:</b>             | Keyword validation string                                | "            | <i>text</i>          | S      |
| <b>:BF:</b>             | Branch flag                                              | "            | <b>yes or no</b>     | S      |
| <b>:J:</b>              | Joint edit flag                                          | "            | <b>yes or no</b>     | S      |
| <b>:LK:</b>             | Locked releases                                          | "            | <b>:R: ...</b>       | S      |
| <b>:Q:</b>              | User-defined keyword                                     | "            | <i>text</i>          | S      |
| <b>:M:</b>              | Module name                                              | "            | <i>text</i>          | S      |



29149

29150

29151

29152

29153

29154

29155

29156

29157

29158

29159

29160

29161

29162

29163

| SCCS File Data Keywords |                              |              |                         |        |
|-------------------------|------------------------------|--------------|-------------------------|--------|
| Keyword                 | Data Item                    | File Section | Value                   | Format |
| <b>:FB:</b>             | Floor boundary               | "            | <b>:R:</b>              | S      |
| <b>:CB:</b>             | Ceiling boundary             | "            | <b>:R:</b>              | S      |
| <b>:Ds:</b>             | Default SID                  | "            | <b>:I:</b>              | S      |
| <b>:ND:</b>             | Null delta flag              | "            | <b>yes or no</b>        | S      |
| <b>:FD:</b>             | File descriptive text        | Comments     | <i>text</i>             | M      |
| <b>:BD:</b>             | Body                         | Body         | <i>text</i>             | M      |
| <b>:GB:</b>             | Gotten body                  | "            | <i>text</i>             | M      |
| <b>:W:</b>              | A form of <i>what</i> string | N / A        | <b>:Z::M:\t:I:</b>      | S      |
| <b>:A:</b>              | A form of <i>what</i> string | N / A        | <b>:Z::Y::M: :I::Z:</b> | S      |
| <b>:Z:</b>              | <i>what</i> string delimiter | N / A        | @ (#)                   | S      |
| <b>:F:</b>              | SCCS filename                | N / A        | <i>text</i>             | S      |
| <b>:PN:</b>             | SCCS file pathname           | N / A        | <i>text</i>             | S      |

29164

\* **:Dt::DT: :I: :D: :T: :P: :DS: :DP:**

29165

\*\* **:R::L::B::S:** if the delta is a branch delta (**:BF:=yes**)

29166

**:R::L:** if the delta is not a branch delta (**:BF:=no**)

29167

\*\*\* The line statistics are capped at 99999. For example, if 100 000 lines were unchanged in a certain revision, **:Lu:** shall produce the value 99999.

29168

**29169 STDERR**

29170

The standard error shall be used only for diagnostic messages.

**29171 OUTPUT FILES**

29172

None.

**29173 EXTENDED DESCRIPTION**

29174

None.

**29175 EXIT STATUS**

29176

The following exit values shall be returned:

29177

0 Successful completion.

29178

&gt;0 An error occurred.

**29179 CONSEQUENCES OF ERRORS**

29180

Default.

**29181 APPLICATION USAGE**

29182

None.

**29183 EXAMPLES**

29184

1. The following example:

29185

`prs -d "User Names for :F: are:\n:UN:" s.file`

29186

might write to standard output:

29187

User Names for s.file are:

29188

xyz

29189

131

29190

abc

29191

2. The following example:

```

29192 prs -d "Delta for pgm :M:: :I: - :D: By :P:" -r s.file
29193 might write to standard output:
29194 Delta for pgm main.c: 3.7 - 77/12/01 By cas
29195 3. As a special case:
29196 prs s.file
29197 might write to standard output:
29198 s.file:
29199 <blank line>
29200 D 1.1 77/12/01 00:00:00 cas 1 000000/00000/00000
29201 MRs:
29202 b178-12345
29203 b179-54321
29204 COMMENTS:
29205 this is the comment line for s.file initial delta
29206 <blank line>
29207 for each delta table entry of the D type. The only option allowed to be used with this
29208 special case is the -a option.

29209 RATIONALE
29210 None.

29211 FUTURE DIRECTIONS
29212 None.

29213 SEE ALSO
29214 admin, delta, get, what

29215 CHANGE HISTORY
29216 First released in Issue 2.

29217 Issue 5
29218 The phrase “in which keyword substitution is followed by a <newline>” is deleted from the end
29219 of the second paragraph of Data Keywords (on page 747).
29220 The interpretation of the YY component of the -c cutoff argument is noted.

29221 Issue 6
29222 The normative text is reworded to emphasize the term “shall” for implementation requirements.
29223 The Open Group Base Resolution bwg2001-007 is applied, updating the table in STDOUT with a
29224 note that line statistics are capped at 99 999 for the :Li:, :Ld:, :Lu:, and :DL: keywords.
29225 The Open Group Interpretation PIN4C.00009 is applied.

```

29226 **NAME**

29227       ps — report process status

29228 **SYNOPSIS**29229 UP XSI   ps [-aA] [-defl] [-G *grouplist*] [-o *format*]...[-p *proclist*] [-t *termlist*]29230       [-U *userlist*] [-g *grouplist*] [-n *namelist*] [-u *userlist*]

29231

29232 **DESCRIPTION**29233       The *ps* utility shall write information about processes, subject to having the appropriate  
29234       privileges to obtain information about those processes.29235       By default, *ps* shall select all processes with the same effective user ID as the current user and the  
29236       same controlling terminal as the invoker.29237 **OPTIONS**29238       The *ps* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
29239       Utility Syntax Guidelines.

29240       The following options shall be supported:

29241       **-a**       Write information for all processes associated with terminals. Implementations  
29242       may omit session leaders from this list.29243       **-A**       Write information for all processes.29244 XSI       **-d**       Write information for all processes, except session leaders.29245 XSI       **-e**       Write information for all processes. (Equivalent to **-A**.)29246 XSI       **-f**       Generate a **full** listing. (See the STDOUT section for the contents of a **full** listing.)29247 XSI       **-g *grouplist*** Write information for processes whose session leaders are given in *grouplist*. The  
29248       application shall ensure that the *grouplist* is a single argument in the form of a  
29249       <blank> or comma-separated list.29250       **-G *grouplist*** Write information for processes whose real group ID numbers are given in  
29251       *grouplist*. The application shall ensure that the *grouplist* is a single argument in the  
29252       form of a <blank> or comma-separated list.29253 XSI       **-l**       Generate a **long** listing. (See STDOUT for the contents of a **long** listing.)29254 XSI       **-n *namelist*** Specify the name of an alternative system *namelist* file in place of the default. The  
29255       name of the default file and the format of a *namelist* file are unspecified.29256       **-o *format***   Write information according to the format specification given in *format*. This is  
29257       fully described in the STDOUT section. Multiple **-o** options can be specified; the  
29258       format specification shall be interpreted as the <space>-separated concatenation of  
29259       all the *format* option-arguments.29260       **-p *proclist*** Write information for processes whose process ID numbers are given in *proclist*.  
29261       The application shall ensure that the *proclist* is a single argument in the form of a  
29262       <blank> or comma-separated list.29263       **-t *termlist*** Write information for processes associated with terminals given in *termlist*. The  
29264       application shall ensure that the *termlist* is a single argument in the form of a  
29265       <blank> or comma-separated list. Terminal identifiers shall be given in an  
29266 XSI       implementation-defined format. On XSI-conformant systems, they shall be given  
29267       in one of two forms: the device's filename (for example, **tty04**) or, if the device's  
29268       filename starts with **tty**, just the identifier following the characters **tty** (for

|           |                              |                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 29269     |                              | example, "04").                                                                                                                                                                                                                                                                                                                                                             |
| 29270 XSI | <b>-u</b> <i>userlist</i>    | Write information for processes whose user ID numbers or login names are given in <i>userlist</i> . The application shall ensure that the <i>userlist</i> is a single argument in the form of a <blank> or comma-separated list. In the listing, the numerical user ID shall be written unless the <b>-f</b> option is used, in which case the login name shall be written. |
| 29271     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29272     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29273     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29274     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29275     | <b>-U</b> <i>userlist</i>    | Write information for processes whose real user ID numbers or login names are given in <i>userlist</i> . The application shall ensure that the <i>userlist</i> is a single argument in the form of a <blank> or comma-separated list.                                                                                                                                       |
| 29276     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29277     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29278     |                              | With the exception of <b>-o</b> <i>format</i> , all of the options shown are used to select processes. If any are specified, the default list shall be ignored and <i>ps</i> shall select the processes represented by the inclusive OR of all the selection-criteria options.                                                                                              |
| 29279     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29280     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29281     | <b>OPERANDS</b>              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29282     |                              | None.                                                                                                                                                                                                                                                                                                                                                                       |
| 29283     | <b>STDIN</b>                 |                                                                                                                                                                                                                                                                                                                                                                             |
| 29284     |                              | Not used.                                                                                                                                                                                                                                                                                                                                                                   |
| 29285     | <b>INPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                             |
| 29286     |                              | None.                                                                                                                                                                                                                                                                                                                                                                       |
| 29287     | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                                                                                                                                                                                                                                             |
| 29288     |                              | The following environment variables shall affect the execution of <i>ps</i> :                                                                                                                                                                                                                                                                                               |
| 29289     | <b>COLUMNS</b>               | Override the system-selected horizontal display line size, used to determine the number of text columns to display. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables for valid values and results when it is unset or null.                                                                                                        |
| 29290     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29291     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29292     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29293     | <b>LANG</b>                  | Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)                                                                       |
| 29294     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29295     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29296     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29297     | <b>LC_ALL</b>                | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                    |
| 29298     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29299     | <b>LC_CTYPE</b>              | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).                                                                                                                                                                                                   |
| 29300     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29301     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29302     | <b>LC_MESSAGES</b>           |                                                                                                                                                                                                                                                                                                                                                                             |
| 29303     |                              | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.                                                                                                                                                                                            |
| 29304     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29305     |                              |                                                                                                                                                                                                                                                                                                                                                                             |
| 29306     | <b>LC_TIME</b>               | Determine the format and contents of the date and time strings displayed.                                                                                                                                                                                                                                                                                                   |
| 29307 XSI | <b>NLSPATH</b>               | Determine the location of message catalogs for the processing of <b>LC_MESSAGES</b> .                                                                                                                                                                                                                                                                                       |
| 29308     | <b>TZ</b>                    | Determine the timezone used to calculate date and time strings displayed. If <b>TZ</b> is unset or null, an unspecified default timezone shall be used.                                                                                                                                                                                                                     |
| 29309     |                              |                                                                                                                                                                                                                                                                                                                                                                             |

29310 **ASYNCHRONOUS EVENTS**

29311 Default.

29312 **STDOUT**29313 When the **-o** option is not specified, the standard output format is unspecified.

29314 XSI On XSI-conformant systems, the output format shall be as follows. The column headings and descriptions of the columns in a *ps* listing are given below. The precise meanings of these fields are implementation-defined. The letters 'f' and 'l' (below) indicate the option (**full** or **long**) that shall cause the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes are listed.

|       |              |       |                                                                                                   |
|-------|--------------|-------|---------------------------------------------------------------------------------------------------|
| 29320 | <b>F</b>     | (l)   | Flags (octal and additive) associated with the process.                                           |
| 29321 | <b>S</b>     | (l)   | The state of the process.                                                                         |
| 29322 | <b>UID</b>   | (f,l) | The user ID number of the process owner; the login name is printed under the <b>-f</b> option.    |
| 29323 |              |       |                                                                                                   |
| 29324 | <b>PID</b>   | (all) | The process ID of the process; it is possible to kill a process if this datum is known.           |
| 29325 |              |       |                                                                                                   |
| 29326 | <b>PPID</b>  | (f,l) | The process ID of the parent process.                                                             |
| 29327 | <b>C</b>     | (f,l) | Processor utilization for scheduling.                                                             |
| 29328 | <b>PRI</b>   | (l)   | The priority of the process; higher numbers mean lower priority.                                  |
| 29329 | <b>NI</b>    | (l)   | Nice value; used in priority computation.                                                         |
| 29330 | <b>ADDR</b>  | (l)   | The address of the process.                                                                       |
| 29331 | <b>SZ</b>    | (l)   | The size in blocks of the core image of the process.                                              |
| 29332 | <b>WCHAN</b> | (l)   | The event for which the process is waiting or sleeping; if blank, the process is running.         |
| 29333 |              |       |                                                                                                   |
| 29334 | <b>STIME</b> | (f)   | Starting time of the process.                                                                     |
| 29335 | <b>TTY</b>   | (all) | The controlling terminal for the process.                                                         |
| 29336 | <b>TIME</b>  | (all) | The cumulative execution time for the process.                                                    |
| 29337 | <b>CMD</b>   | (all) | The command name; the full command name and its arguments are written under the <b>-f</b> option. |
| 29338 |              |       |                                                                                                   |

29339 A process that has exited and has a parent, but has not yet been waited for by the parent, shall be marked **defunct**.

29341 Under the option **-f**, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the option **-f**, is written in square brackets.

29344 The **-o** option allows the output format to be specified under user control.

29345 The application shall ensure that the format specification is a list of names presented as a single argument, <blank> or comma-separated. Each variable has a default header. The default header can be overridden by appending an equals sign and the new text of the header. The rest of the characters in the argument shall be used as the header text. The fields specified shall be written in the order specified on the command line, and should be arranged in columns in the output. The field widths shall be selected by the system to be at least as wide as the header text (default or overridden value). If the header text is null, such as **-o user=**, the field width shall be at least as wide as the default header text. If all header text fields are null, no header line shall be written.

29354 The following names are recognized in the POSIX locale:

|       |                                                                                                  |                                                                                                                   |
|-------|--------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| 29355 | <b>ruser</b>                                                                                     | The real user ID of the process. This shall be the textual user ID, if it can be obtained                         |
| 29356 |                                                                                                  | and the field width permits, or a decimal representation otherwise.                                               |
| 29357 | <b>user</b>                                                                                      | The effective user ID of the process. This shall be the textual user ID, if it can be                             |
| 29358 |                                                                                                  | obtained and the field width permits, or a decimal representation otherwise.                                      |
| 29359 | <b>rgroup</b>                                                                                    | The real group ID of the process. This shall be the textual group ID, if it can be obtained                       |
| 29360 |                                                                                                  | and the field width permits, or a decimal representation otherwise.                                               |
| 29361 | <b>group</b>                                                                                     | The effective group ID of the process. This shall be the textual group ID, if it can be                           |
| 29362 |                                                                                                  | obtained and the field width permits, or a decimal representation otherwise.                                      |
| 29363 | <b>pid</b>                                                                                       | The decimal value of the process ID.                                                                              |
| 29364 | <b>ppid</b>                                                                                      | The decimal value of the parent process ID.                                                                       |
| 29365 | <b>pgid</b>                                                                                      | The decimal value of the process group ID.                                                                        |
| 29366 | <b>pcpu</b>                                                                                      | The ratio of CPU time used recently to CPU time available in the same period,                                     |
| 29367 |                                                                                                  | expressed as a percentage. The meaning of “recently” in this context is unspecified. The                          |
| 29368 |                                                                                                  | CPU time available is determined in an unspecified manner.                                                        |
| 29369 | <b>vsz</b>                                                                                       | The size of the process in (virtual) memory in 1 024 byte units as a decimal integer.                             |
| 29370 | <b>nice</b>                                                                                      | The decimal value of the nice value of the process; see <i>nice</i> .                                             |
| 29371 | <b>etime</b>                                                                                     | In the POSIX locale, the elapsed time since the process was started, in the form:                                 |
| 29372 |                                                                                                  | [ [ <i>dd</i> –] <i>hh</i> : ] <i>mm</i> : <i>ss</i>                                                              |
| 29373 |                                                                                                  | where <i>dd</i> shall represent the number of days, <i>hh</i> the number of hours, <i>mm</i> the number           |
| 29374 |                                                                                                  | of minutes, and <i>ss</i> the number of seconds. The <i>dd</i> field shall be a decimal integer. The              |
| 29375 |                                                                                                  | <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be two-digit decimal integers padded on the left with zeros.   |
| 29376 | <b>time</b>                                                                                      | In the POSIX locale, the cumulative CPU time of the process in the form:                                          |
| 29377 |                                                                                                  | [ [ <i>dd</i> –] <i>hh</i> : <i>mm</i> : <i>ss</i>                                                                |
| 29378 |                                                                                                  | The <i>dd</i> , <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be as described in the <b>etime</b> specifier. |
| 29379 | <b>tty</b>                                                                                       | The name of the controlling terminal of the process (if any) in the same format used by                           |
| 29380 |                                                                                                  | the <i>who</i> utility.                                                                                           |
| 29381 | <b>comm</b>                                                                                      | The name of the command being executed ( <i>argv</i> [0] value) as a string.                                      |
| 29382 | <b>args</b>                                                                                      | The command with all its arguments as a string. The implementation may truncate this                              |
| 29383 |                                                                                                  | value to the field width; it is implementation-defined whether any further truncation                             |
| 29384 |                                                                                                  | occurs. It is unspecified whether the string represented is a version of the argument list                        |
| 29385 |                                                                                                  | as it was passed to the command when it started, or is a version of the arguments as                              |
| 29386 |                                                                                                  | they may have been modified by the application. Applications cannot depend on being                               |
| 29387 |                                                                                                  | able to modify their argument list and having that modification be reflected in the                               |
| 29388 |                                                                                                  | output of <i>ps</i> .                                                                                             |
| 29389 | Any field need not be meaningful in all implementations. In such a case a hyphen (‘–’) should    |                                                                                                                   |
| 29390 | be output in place of the field value.                                                           |                                                                                                                   |
| 29391 | Only <b>comm</b> and <b>args</b> shall be allowed to contain <blank>s; all others shall not. Any |                                                                                                                   |
| 29392 | implementation-defined variables shall be specified in the system documentation along with the   |                                                                                                                   |
| 29393 | default header and indicating whether the field may contain <blank>s.                            |                                                                                                                   |
| 29394 | The following table specifies the default header to be used in the POSIX locale corresponding to |                                                                                                                   |
| 29395 | each format specifier.                                                                           |                                                                                                                   |

Table 4-17 Variable Names and Default Headers in *ps*

| Format Specifier | Default Header | Format Specifier | Default Header |
|------------------|----------------|------------------|----------------|
| <b>args</b>      | <b>COMMAND</b> | <b>ppid</b>      | <b>PPID</b>    |
| <b>comm</b>      | <b>COMMAND</b> | <b>rgroup</b>    | <b>RGROUP</b>  |
| <b>etime</b>     | <b>ELAPSED</b> | <b>ruser</b>     | <b>RUSER</b>   |
| <b>group</b>     | <b>GROUP</b>   | <b>time</b>      | <b>TIME</b>    |
| <b>nice</b>      | <b>NI</b>      | <b>tty</b>       | <b>TT</b>      |
| <b>pcpu</b>      | <b>%CPU</b>    | <b>user</b>      | <b>USER</b>    |
| <b>pgid</b>      | <b>PGID</b>    | <b>vsz</b>       | <b>VSZ</b>     |
| <b>pid</b>       | <b>PID</b>     |                  |                |

**STDERR**

The standard error shall be used only for diagnostic messages.

**OUTPUT FILES**

None.

**EXTENDED DESCRIPTION**

None.

**EXIT STATUS**

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

**CONSEQUENCES OF ERRORS**

Default.

**APPLICATION USAGE**

Things can change while *ps* is running; the snapshot it gives is only true for an instant, and might not be accurate by the time it is displayed.

The **args** format specifier is allowed to produce a truncated version of the command arguments. In some implementations, this information is no longer available when the *ps* utility is executed.

If the field width is too narrow to display a textual ID, the system may use a numeric version. Normally, the system would be expected to choose large enough field widths, but if a large number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on one line. One way to ensure adequate width for the textual IDs is to override the default header for a field to make it larger than most or all user or group names.

There is no special quoting mechanism for header text. The header text is the rest of the argument. If multiple header changes are needed, multiple **-o** options can be used, such as:

```
ps -o "user=User Name" -o pid=Process\ ID
```

On some implementations, especially multi-level secure systems, *ps* may be severely restricted and produce information only about child processes owned by the user.

**EXAMPLES**

The command:

```
ps -o user,pid,ppid=MOM -o args
```

writes at least the following in the POSIX locale:

```
USER PID MOM COMMAND
helene 34 12 ps -o uid,pid,ppid=MOM -o args
```

29439 The contents of the **COMMAND** field need not be the same in all implementations, due to  
29440 possible truncation.

#### 29441 **RATIONALE**

29442 There is very little commonality between BSD and System V implementations of *ps*. Many  
29443 options conflict or have subtly different usages. The standard developers attempted to select a  
29444 set of options for the base standard that were useful on a wide range of systems and selected  
29445 options that either can be implemented on both BSD and System V-based systems without  
29446 breaking the current implementations or where the options are sufficiently similar that any  
29447 changes would not be unduly problematic for users or implementors.

29448 It is recognized that on some implementations, especially multi-level secure systems, *ps* may be  
29449 nearly useless. The default output has therefore been chosen such that it does not break  
29450 historical implementations and also is likely to provide at least some useful information on most  
29451 systems.

29452 The major change is the addition of the format specification capability. The motivation for this  
29453 invention is to provide a mechanism for users to access a wider range of system information, if  
29454 the system permits it, in a portable manner. The fields chosen to appear in this volume of  
29455 IEEE Std 1003.1-2001 were arrived at after considering what concepts were likely to be both  
29456 reasonably useful to the “average” user and had a reasonable chance of being implemented on a  
29457 wide range of systems. Again it is recognized that not all systems are able to provide all the  
29458 information and, conversely, some may wish to provide more. It is hoped that the approach  
29459 adopted will be sufficiently flexible and extensible to accommodate most systems.  
29460 Implementations may be expected to introduce new format specifiers.

29461 The default output should consist of a short listing containing the process ID, terminal name,  
29462 cumulative execution time, and command name of each process.

29463 The preference of the standard developers would have been to make the format specification an  
29464 operand of the *ps* command. Unfortunately, BSD usage precluded this.

29465 At one time a format was included to display the environment array of the process. This was  
29466 deleted because there is no portable way to display it.

29467 The **-A** option is equivalent to the BSD **-g** and the SVID **-e**. Because the two systems differed, a  
29468 mnemonic compromise was selected.

29469 The **-a** option is described with some optional behavior because the SVID omits session leaders,  
29470 but BSD does not.

29471 In an early proposal, format specifiers appeared for priority and start time. The former was not  
29472 defined adequately in this volume of IEEE Std 1003.1-2001 and was removed in deference to the  
29473 defined nice value; the latter because elapsed time was considered to be more useful.

29474 In a new BSD version of *ps*, a **-O** option can be used to write all of the default information,  
29475 followed by additional format specifiers. This was not adopted because the default output is  
29476 implementation-defined. Nevertheless, this is a useful option that should be reserved for that  
29477 purpose. In the **-o** option for the POSIX Shell and Utilities *ps*, the format is the concatenation of  
29478 each **-o**. Therefore, the user can have an alias or function that defines the beginning of their  
29479 desired format and add more fields to the end of the output in certain cases where that would be  
29480 useful.

29481 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
29482 require that they all use the same format.

29483 The **pcpu** field indicates that the CPU time available is determined in an unspecified manner.  
29484 This is because it is difficult to express an algorithm that is useful across all possible machine



29485 architectures. Historical counterparts to this value have attempted to show percentage of use in  
29486 the recent past, such as the preceding minute. Frequently, these values for all processes did not  
29487 add up to 100%. Implementations are encouraged to provide data in this field to users that will  
29488 help them identify processes currently affecting the performance of the system.

29489 **FUTURE DIRECTIONS**

29490 None.

29491 **SEE ALSO**

29492 *kill, nice, renice*

29493 **CHANGE HISTORY**

29494 First released in Issue 2.

29495 **Issue 6**

29496 This utility is marked as part of the User Portability Utilities option.

29497 The normative text is reworded to avoid use of the term “must” for application requirements.

29498 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

29499 **NAME**

29500           pwd — return working directory name

29501 **SYNOPSIS**

29502           pwd [-L | -P ]

29503 **DESCRIPTION**

29504           The *pwd* utility shall write to standard output an absolute pathname of the current working  
 29505           directory, which does not contain the filenames dot or dot-dot.

29506 **OPTIONS**

29507           The *pwd* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 29508           12.2, Utility Syntax Guidelines.

29509           The following options shall be supported by the implementation:

29510           **-L**           If the *PWD* environment variable contains an absolute pathname of the current  
 29511                            directory that does not contain the filenames dot or dot-dot, *pwd* shall write this  
 29512                            pathname to standard output. Otherwise, the **-L** option shall behave as the **-P**  
 29513                            option.

29514           **-P**           The absolute pathname written shall not contain filenames that, in the context of  
 29515                            the pathname, refer to files of type symbolic link.

29516           If both **-L** and **-P** are specified, the last one shall apply. If neither **-L** nor **-P** is specified, the *pwd*  
 29517           utility shall behave as if **-L** had been specified.

29518 **OPERANDS**

29519           None.

29520 **STDIN**

29521           Not used.

29522 **INPUT FILES**

29523           None.

29524 **ENVIRONMENT VARIABLES**

29525           The following environment variables shall affect the execution of *pwd*:

29526           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
 29527                            (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 29528                            Internationalization Variables for the precedence of internationalization variables  
 29529                            used to determine the values of locale categories.)

29530           **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 29531                            internationalization variables.

29532           **LC\_MESSAGES**

29533                           Determine the locale that should be used to affect the format and contents of  
 29534                            diagnostic messages written to standard error.

29535 **XS1**           **NLS\_PATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

29536           **PWD**           If the **-P** option is in effect, this variable shall be set to an absolute pathname of the  
 29537                            current working directory that does not contain any components that specify  
 29538                            symbolic links, does not contain any components that are dot, and does not  
 29539                            contain any components that are dot-dot. If an application sets or unsets the value  
 29540                            of *PWD*, the behavior of *pwd* is unspecified.

29541 **ASYNCHRONOUS EVENTS**

29542 Default.

29543 **STDOUT**29544 The *pwd* utility output is an absolute pathname of the current working directory:

29545 "%s\n", &lt;directory pathname&gt;

29546 **STDERR**

29547 The standard error shall be used only for diagnostic messages.

29548 **OUTPUT FILES**

29549 None.

29550 **EXTENDED DESCRIPTION**

29551 None.

29552 **EXIT STATUS**

29553 The following exit values shall be returned:

29554 0 Successful completion.

29555 &gt;0 An error occurred.

29556 **CONSEQUENCES OF ERRORS**29557 If an error is detected, output shall not be written to standard output, a diagnostic message shall  
29558 be written to standard error, and the exit status is not zero.29559 **APPLICATION USAGE**

29560 None.

29561 **EXAMPLES**

29562 None.

29563 **RATIONALE**29564 Some implementations have historically provided *pwd* as a shell special built-in command.

29565 In most utilities, if an error occurs, partial output may be written to standard output. This does  
29566 not happen in historical implementations of *pwd*. Because *pwd* is frequently used in historical  
29567 shell scripts without checking the exit status, it is important that the historical behavior is  
29568 required here; therefore, the CONSEQUENCES OF ERRORS section specifically disallows any  
29569 partial output being written to standard output.

29570 **FUTURE DIRECTIONS**

29571 None.

29572 **SEE ALSO**29573 *cd*, the System Interfaces volume of IEEE Std 1003.1-2001, *getcwd()*29574 **CHANGE HISTORY**

29575 First released in Issue 2.

29576 **Issue 6**

29577 The **-P** and **-L** options are added to describe actions relating to symbolic links as specified in the  
29578 IEEE P1003.2b draft standard.

29579 **NAME**

29580 qalter — alter batch job

29581 **SYNOPSIS**

```
29582 BE qalter [-a date_time] [-A account_string] [-c interval] [-e path_name]
29583 [-h hold_list] [-j join_list] [-k keep_list] [-l resource_list]
29584 [-m mail_options] [-M mail_list] [-N name] [-o path_name]
29585 [-p priority] [-r y|n] [-S path_name_list] [-u user_list]
29586 job_identifier ...
29587
```

29588 **DESCRIPTION**

29589 The attributes of a batch job are altered by a request to the batch server that manages the batch  
 29590 job. The *qalter* utility is a user-accessible batch client that requests the alteration of the attributes  
 29591 of one or more batch jobs.

29592 The *qalter* utility shall alter the attributes of those batch jobs, and only those batch jobs, for which  
 29593 a batch *job\_identifier* is presented to the utility.

29594 The *qalter* utility shall alter the attributes of batch jobs in the order in which the batch  
 29595 *job\_identifiers* are presented to the utility.

29596 If the *qalter* utility fails to process a batch *job\_identifier* successfully, the utility shall proceed to  
 29597 process the remaining batch *job\_identifiers*, if any.

29598 For each batch *job\_identifier* for which the *qalter* utility succeeds, each attribute of the identified  
 29599 batch job shall be altered as indicated by all the options presented to the utility.

29600 For each identified batch job for which the *qalter* utility fails, the utility shall not alter any  
 29601 attribute of the batch job.

29602 For each batch job that the *qalter* utility processes, the utility shall not modify any attribute other  
 29603 than those required by the options and option-arguments presented to the utility.

29604 The *qalter* utility shall alter batch jobs by sending a *Modify Job Request* to the batch server that  
 29605 manages each batch job. At the time the *qalter* utility exits, it shall have modified the batch job  
 29606 corresponding to each successfully processed batch *job\_identifier*. An attempt to alter the  
 29607 attributes of a batch job in the RUNNING state is implementation-defined.

29608 **OPTIONS**

29609 The *qalter* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 29610 12.2, Utility Syntax Guidelines.

29611 The following options shall be supported by the implementation:

29612 **-a date\_time** Redefine the time at which the batch job becomes eligible for execution.

29613 The *date\_time* argument shall be in the same form and represent the same time as  
 29614 for the *touch* utility. The time so represented shall be set into the *Execution\_Time*  
 29615 attribute of the batch job. If the time specified is earlier than the current time, the  
 29616 **-a** option shall have no effect.

29617 **-A account\_string**

29618 Redefine the account to which the resource consumption of the batch job should be  
 29619 charged.

29620 The syntax of the *account\_string* option-argument is unspecified.

29621 The *qalter* utility shall set the *Account\_Name* attribute of the batch job to the value  
 29622 of the *account\_string* option-argument.

29623       **-c interval**     Redefine whether the batch job should be checkpointed, and if so, how often.

29624                     The *qalter* utility shall accept a value for the interval option-argument that is one of

29625                     the following:

29626               n             No checkpointing is to be performed on the batch job

29627                     (NO\_CHECKPOINT).

29628               s             Checkpointing is to be performed only when the batch server is shut

29629                     down (CHECKPOINT\_AT\_SHUTDOWN).

29630               c             Automatic periodic checkpointing is to be performed at the

29631                     *Minimum\_Cpu\_Interval* attribute of the batch queue, in units of CPU

29632                     minutes (CHECKPOINT\_AT\_MIN\_CPU\_INTERVAL).

29633               c=*minutes*   Automatic periodic checkpointing is to be performed every *minutes*

29634                     of CPU time, or every *Minimum\_Cpu\_Interval* minutes, whichever is

29635                     greater. The *minutes* argument shall conform to the syntax for

29636                     unsigned integers and shall be greater than zero.

29637                     An implementation may define other checkpoint intervals. The conformance

29638                     document for an implementation shall describe any alternative checkpoint

29639                     intervals, how they are specified, their internal behavior, and how they affect the

29640                     behavior of the utility.

29641                     The *qalter* utility shall set the *Checkpoint* attribute of the batch job to the value of the

29642                     *interval* option-argument.

29643       **-e path\_name**       Redefine the path to be used for the standard error stream of the batch job. |

29644                     The *qalter* utility shall accept a *path\_name* option-argument that conforms to the |

29645                     syntax of the *path\_name* element defined in the System Interfaces volume of

29646                     IEEE Std 1003.1-2001, which can be preceded by a host name element of the form

29647                     *hostname*::.

29648                     If the *path\_name* option-argument constitutes an absolute pathname, the *qalter*

29649                     utility shall set the *Error\_Path* attribute of the batch job to the value of the

29650                     *path\_name* option-argument, including the host name element, if present.

29651                     If the *path\_name* option-argument constitutes a relative pathname and no host

29652                     name element is specified, the *qalter* utility shall set the *Error\_Path* attribute of the

29653                     batch job to the value of the absolute pathname derived by expanding the

29654                     *path\_name* option-argument relative to the current directory of the process that

29655                     executes the *qalter* utility.

29656                     If the *path\_name* option-argument constitutes a relative pathname and a host name

29657                     element is specified, the *qalter* utility shall set the *Error\_Path* attribute of the batch

29658                     job to the value of the option-argument without expansion.

29659                     If the *path\_name* option-argument does not include a host name element, the *qalter*

29660                     utility shall prefix the pathname in the *Error\_Path* attribute with *hostname*::, where

29661                     *hostname* is the name of the host upon which the *qalter* utility is being executed.

29662                     29663       **-h hold\_list**     Redefine the types of holds, if any, on the batch job. The *qalter* **-h** option shall

29664                     accept a value for the *hold\_list* option-argument that is a string of alphanumeric

29665                     characters in the portable character set.

29666                     The *qalter* utility shall accept a value for the *hold\_list* option-argument that is a

29667                     string of one or more of the characters 'u', 's', or 'o', or the single character

29668 'n'. For each unique character in the *hold\_list* option-argument, the *qalter* utility  
 29669 shall add a value to the *Hold\_Types* attribute of the batch job as follows, each  
 29670 representing a different hold type:

29671 u USER  
 29672 s SYSTEM  
 29673 o OPERATOR

29674 If any of these characters are duplicated in the *hold\_list* option-argument, the  
 29675 duplicates shall be ignored. An existing *Hold\_Types* attribute can be cleared by the  
 29676 hold type:

29677 n NO\_HOLD

29678 The *qalter* utility shall consider it an error if any hold type other than 'n' is  
 29679 combined with hold type 'n'. Strictly conforming applications shall not repeat  
 29680 any of the characters 'u', 's', 'o', or 'n' within the *hold\_list* option-argument.  
 29681 The *qalter* utility shall permit the repetition of characters, but shall not assign  
 29682 additional meaning to the repeated characters. An implementation may define  
 29683 other hold types. The conformance document for an implementation shall describe  
 29684 any additional hold types, how they are specified, their internal behavior, and how  
 29685 they affect the behavior of the utility.

29686 **-j join\_list** Redefine which streams of the batch job are to be merged. The *qalter* **-j** option shall  
 29687 accept a value for the *join\_list* option-argument that is a string of alphanumeric  
 29688 characters in the portable character set.

29689 The *qalter* utility shall accept a *join\_list* option-argument that consists of one or  
 29690 more of the characters 'e' and 'o', or the single character 'n'.

29691 All of the other batch job output streams specified shall be merged into the output  
 29692 stream represented by the character listed first in the *join\_list* option-argument.

29693 For each unique character in the *join\_list* option-argument, the *qalter* utility shall  
 29694 add a value to the *Join\_Path* attribute of the batch job as follows, each representing  
 29695 a different batch job stream to join:

29696 e The standard error of the batch job (JOIN\_STD\_ERROR).  
 29697 o The standard output of the batch job (JOIN\_STD\_OUTPUT).

29698 An existing *Join\_Path* attribute can be cleared by the join type:

29699 n NO\_JOIN

29700 If 'n' is specified, then no files are joined. The *qalter* utility shall consider it an  
 29701 error if any join type other than 'n' is combined with join type 'n'.

29702 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or  
 29703 'n' within the *join\_list* option-argument. The *qalter* utility shall permit the  
 29704 repetition of characters, but shall not assign additional meaning to the repeated  
 29705 characters.

29706 An implementation may define other join types. The conformance document for an  
 29707 implementation shall describe any additional batch job streams, how they are  
 29708 specified, their internal behavior, and how they affect the behavior of the utility.

29709 **-k keep\_list** Redefine which output of the batch job to retain on the execution host.

29710 The *qalter* **-k** option shall accept a value for the *keep\_list* option-argument that is a  
 29711 string of alphanumeric characters in the portable character set.

29712 The *qalter* utility shall accept a *keep\_list* option-argument that consists of one or  
 29713 more of the characters 'e' and 'o', or the single character 'n'.

29714 For each unique character in the *keep\_list* option-argument, the *qalter* utility shall  
 29715 add a value to the *Keep\_Files* attribute of the batch job as follows, each representing  
 29716 a different batch job stream to keep:

- 29717 e The standard error of the batch job (KEEP\_STD\_ERROR).
- 29718 o The standard output of the batch job (KEEP\_STD\_OUTPUT).

29719 If both 'e' and 'o' are specified, then both files are retained. An existing  
 29720 *Keep\_Files* attribute can be cleared by the keep type:

29721 n NO\_KEEP

29722 If 'n' is specified, then no files are retained. The *qalter* utility shall consider it an  
 29723 error if any keep type other than 'n' is combined with keep type 'n'.

29724 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or  
 29725 'n' within the *keep\_list* option-argument. The *qalter* utility shall permit the  
 29726 repetition of characters, but shall not assign additional meaning to the repeated  
 29727 characters. An implementation may define other keep types. The conformance  
 29728 document for an implementation shall describe any additional keep types, how  
 29729 they are specified, their internal behavior, and how they affect the behavior of the  
 29730 utility.

29731 **-l resource\_list**

29732 Redefine the resources that are allowed or required by the batch job.

29733 The *qalter* utility shall accept a *resource\_list* option-argument that conforms to the  
 29734 following syntax:

29735 resource=value[, , resource=value, , ...]

29736 The *qalter* utility shall set one entry in the value of the *Resource\_List* attribute of the  
 29737 batch job for each resource listed in the *resource\_list* option-argument.

29738 Because the list of supported resource names might vary by batch server, the *qalter*  
 29739 utility shall rely on the batch server to validate the resource names and associated  
 29740 values. See Section 3.3.3 (on page 123) for a means of removing *keyword=value* (and  
 29741 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

29742 **-m mail\_options**

29743 Redefine the points in the execution of the batch job at which the batch server is to  
 29744 send mail about a change in the state of the batch job.

29745 The *qalter* **-m** option shall accept a value for the *mail\_options* option-argument that  
 29746 is a string of alphanumeric characters in the portable character set.

29747 The *qalter* utility shall accept a value for the *mail\_options* option-argument that is a  
 29748 string of one or more of the characters 'e', 'b', and 'a', or the single character  
 29749 'n'. For each unique character in the *mail\_options* option-argument, the *qalter*  
 29750 utility shall add a value to the *Mail\_Users* attribute of the batch job as follows, each  
 29751 representing a different time during the life of a batch job at which to send mail:

- 29752 e MAIL\_AT\_EXIT

|       |           |                  |                                                                                                        |
|-------|-----------|------------------|--------------------------------------------------------------------------------------------------------|
| 29753 |           | b                | MAIL_AT_BEGINNING                                                                                      |
| 29754 |           | a                | MAIL_AT_ABORT                                                                                          |
| 29755 |           |                  | If any of these characters are duplicated in the <i>mail_options</i> option-argument, the              |
| 29756 |           |                  | duplicates shall be ignored.                                                                           |
| 29757 |           |                  | An existing <i>Mail_Points</i> attribute can be cleared by the mail type:                              |
| 29758 |           | n                | NO_MAIL                                                                                                |
| 29759 |           |                  | If 'n' is specified, then mail is not sent. The <i>qalter</i> utility shall consider it an error       |
| 29760 |           |                  | if any mail type other than 'n' is combined with mail type 'n'. Strictly                               |
| 29761 |           |                  | conforming applications shall not repeat any of the characters 'e', 'b', 'a', or                       |
| 29762 |           |                  | 'n' within the <i>mail_options</i> option-argument. The <i>qalter</i> utility shall permit the         |
| 29763 |           |                  | repetition of characters but shall not assign additional meaning to the repeated                       |
| 29764 |           |                  | characters.                                                                                            |
| 29765 |           |                  | An implementation may define other mail types. The conformance document for                            |
| 29766 |           |                  | an implementation shall describe any additional mail types, how they are                               |
| 29767 |           |                  | specified, their internal behavior, and how they affect the behavior of the utility.                   |
| 29768 | <b>-M</b> | <i>mail_list</i> | Redefine the list of users to which the batch server that executes the batch job is to                 |
| 29769 |           |                  | send mail, if the batch server sends mail about the batch job.                                         |
| 29770 |           |                  | The syntax of the <i>mail_list</i> option-argument is unspecified. If the implementation               |
| 29771 |           |                  | of the <i>qalter</i> utility uses a name service to locate users, the utility shall accept the         |
| 29772 |           |                  | syntax used by the name service.                                                                       |
| 29773 |           |                  | If the implementation of the <i>qalter</i> utility does not use a name service to locate               |
| 29774 |           |                  | users, the implementation shall accept the following syntax for user names:                            |
| 29775 |           |                  | <i>mail_address</i> [, , <i>mail_address</i> , , ...]                                                  |
| 29776 |           |                  | The interpretation of <i>mail_address</i> is implementation-defined.                                   |
| 29777 |           |                  | The <i>qalter</i> utility shall set the <i>Mail_Users</i> attribute of the batch job to the value of   |
| 29778 |           |                  | the <i>mail_list</i> option-argument.                                                                  |
| 29779 | <b>-N</b> | <i>name</i>      | Redefine the name of the batch job.                                                                    |
| 29780 |           |                  | The <i>qalter</i> <b>-N</b> option shall accept a value for the <i>name</i> option-argument that is a  |
| 29781 |           |                  | string of up to 15 alphanumeric characters in the portable character set where the                     |
| 29782 |           |                  | first character is alphabetic.                                                                         |
| 29783 |           |                  | The syntax of the <i>name</i> option-argument is unspecified.                                          |
| 29784 |           |                  | The <i>qalter</i> utility shall set the <i>Job_Name</i> attribute of the batch job to the value of the |
| 29785 |           |                  | <i>name</i> option-argument.                                                                           |
| 29786 | <b>-o</b> | <i>path_name</i> |                                                                                                        |
| 29787 |           |                  | Redefine the path for the standard output of the batch job.                                            |
| 29788 |           |                  | The <i>qalter</i> utility shall accept a <i>path_name</i> option-argument that conforms to the         |
| 29789 |           |                  | syntax of the <i>path_name</i> element defined in the System Interfaces volume of                      |
| 29790 |           |                  | IEEE Std 1003.1-2001, which can be preceded by a host name element of the form                         |
| 29791 |           |                  | <i>hostname</i> ::.                                                                                    |
| 29792 |           |                  | If the <i>path_name</i> option-argument constitutes an absolute pathname, the <i>qalter</i>            |
| 29793 |           |                  | utility shall set the <i>Output_Path</i> attribute of the batch job to the value of the                |
| 29794 |           |                  | <i>path_name</i> option-argument.                                                                      |



|       |                          |                                                                                                         |
|-------|--------------------------|---------------------------------------------------------------------------------------------------------|
| 29795 |                          | If the <i>path_name</i> option-argument constitutes a relative pathname and no host                     |
| 29796 |                          | name element is specified, the <i>qalter</i> utility shall set the <i>Output_Path</i> attribute of the  |
| 29797 |                          | batch job to the absolute pathname derived by expanding the <i>path_name</i> option-                    |
| 29798 |                          | argument relative to the current directory of the process that executes the <i>qalter</i>               |
| 29799 |                          | utility.                                                                                                |
| 29800 |                          | If the <i>path_name</i> option-argument constitutes a relative pathname and a host name                 |
| 29801 |                          | element is specified, the <i>qalter</i> utility shall set the <i>Output_Path</i> attribute of the batch |
| 29802 |                          | job to the value of the <i>path_name</i> option-argument without any expansion of the                   |
| 29803 |                          | pathname.                                                                                               |
| 29804 |                          | If the <i>path_name</i> option-argument does not include a host name element, the <i>qalter</i>         |
| 29805 |                          | utility shall prefix the pathname in the <i>Output_Path</i> attribute with <i>hostname</i> :, where     |
| 29806 |                          | <i>hostname</i> is the name of the host upon which the <i>qalter</i> utility is being executed.         |
| 29807 | <b>-p priority</b>       | Redefine the priority of the batch job.                                                                 |
| 29808 |                          | The <i>qalter</i> utility shall accept a value for the priority option-argument that                    |
| 29809 |                          | conforms to the syntax for signed decimal integers, and which is not less than                          |
| 29810 |                          | -1 024 and not greater than 1 023.                                                                      |
| 29811 |                          | The <i>qalter</i> utility shall set the <i>Priority</i> attribute of the batch job to the value of the  |
| 29812 |                          | <i>priority</i> option-argument.                                                                        |
| 29813 | <b>-r y   n</b>          | Redefine whether the batch job is rerunnable.                                                           |
| 29814 |                          | If the value of the option-argument is 'y', the <i>qalter</i> utility shall set the <i>Rerunable</i>    |
| 29815 |                          | attribute of the batch job to TRUE.                                                                     |
| 29816 |                          | If the value of the option-argument is 'n', the <i>qalter</i> utility shall set the <i>Rerunable</i>    |
| 29817 |                          | attribute of the batch job to FALSE.                                                                    |
| 29818 |                          | The <i>qalter</i> utility shall consider it an error if any character other than 'y' or 'n' is          |
| 29819 |                          | specified in the option-argument.                                                                       |
| 29820 | <b>-S path_name_list</b> |                                                                                                         |
| 29821 |                          | Redefine the shell that interprets the script at the destination system.                                |
| 29822 |                          | The <i>qalter</i> utility shall accept a <i>path_name_list</i> option-argument that conforms to         |
| 29823 |                          | the following syntax:                                                                                   |
| 29824 |                          | pathname[@host] [, pathname[@host] , ...]                                                               |
| 29825 |                          | The <i>qalter</i> utility shall accept only one pathname that is missing a corresponding                |
| 29826 |                          | host name. The <i>qalter</i> utility shall allow only one pathname per named host.                      |
| 29827 |                          | The <i>qalter</i> utility shall add a value to the <i>Shell_Path_List</i> attribute of the batch job    |
| 29828 |                          | for each entry in the <i>path_name_list</i> option-argument. See Section 3.3.3 (on page                 |
| 29829 |                          | 123) for a means of removing <i>keyword=value</i> (and <i>value@keyword</i> ) pairs and other           |
| 29830 |                          | general rules for list-oriented batch job attributes.                                                   |
| 29831 | <b>-u user_list</b>      | Redefine the user name under which the batch job is to run at the destination                           |
| 29832 |                          | system.                                                                                                 |
| 29833 |                          | The <i>qalter</i> utility shall accept a <i>user_list</i> option-argument that conforms to the          |
| 29834 |                          | following syntax:                                                                                       |
| 29835 |                          | username[@host] [, , username[@host] , , ...]                                                           |
| 29836 |                          | The <i>qalter</i> utility shall accept only one user name that is missing a corresponding               |
| 29837 |                          | host name. The <i>qalter</i> utility shall accept only one user name per named host.                    |

29838 The *qalter* utility shall add a value to the *User\_List* attribute of the batch job for each  
 29839 entry in the *user\_list* option-argument. See Section 3.3.3 (on page 123) for a means  
 29840 of removing *keyword=value* (and *value@keyword*) pairs and other general rules for  
 29841 list-oriented batch job attributes.

#### 29842 OPERANDS

29843 The *qalter* utility shall accept one or more operands that conform to the syntax for a batch  
 29844 *job\_identifier* (see Section 3.3.1 (on page 122)).

#### 29845 STDIN

29846 Not used.

#### 29847 INPUT FILES

29848 None.

#### 29849 ENVIRONMENT VARIABLES

29850 The following environment variables shall affect the execution of *qalter*:

29851 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 29852 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 29853 Internationalization Variables for the precedence of internationalization variables  
 29854 used to determine the values of locale categories.)

29855 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 29856 internationalization variables.

29857 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 29858 characters (for example, single-byte as opposed to multi-byte characters in  
 29859 arguments).

#### 29860 LC\_MESSAGES

29861 Determine the locale that should be used to affect the format and contents of  
 29862 diagnostic messages written to standard error.

29863 *LOGNAME* Determine the login name of the user.

29864 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is  
 29865 unset or null, an unspecified default timezone shall be used.

#### 29866 ASYNCHRONOUS EVENTS

29867 Default.

#### 29868 STDOUT

29869 None.

#### 29870 STDERR

29871 The standard error shall be used only for diagnostic messages.

#### 29872 OUTPUT FILES

29873 None.

#### 29874 EXTENDED DESCRIPTION

29875 None.

#### 29876 EXIT STATUS

29877 The following exit values shall be returned:

29878 0 Successful completion.

29879 >0 An error occurred.

29880 **CONSEQUENCES OF ERRORS**

29881 In addition to the default behavior, the *qalter* utility shall not be required to write a diagnostic  
 29882 message to standard error when the error reply received from a batch server indicates that the  
 29883 batch *job\_identifier* does not exist on the server. Whether or not the *qalter* utility attempts to  
 29884 locate the batch job on other batch servers is implementation-defined.

29885 **APPLICATION USAGE**

29886 None.

29887 **EXAMPLES**

29888 None.

29889 **RATIONALE**

29890 The *qalter* utility allows users to change the attributes of a batch job.

29891 As a means of altering a queued job, the *qalter* utility is superior to deleting and requeuing the  
 29892 batch job insofar as an altered job retains its place in the queue with some traditional selection  
 29893 algorithms. In addition, the *qalter* utility is both shorter and simpler than a sequence of *qdel* and  
 29894 *qsub* utilities.

29895 The result of an attempt on the part of a user to alter a batch job in a RUNNING state is  
 29896 implementation-defined because a batch job in the RUNNING state will already have opened its  
 29897 output files and otherwise performed any actions indicated by the options in effect at the time  
 29898 the batch job began execution.

29899 The options processed by the *qalter* utility are identical to those of the *qsub* utility, with a few  
 29900 exceptions: **-V**, **-v**, and **-q**. The **-V** and **-v** are inappropriate for the *qalter* utility, since they  
 29901 capture potentially transient environment information from the submitting process. The **-q**  
 29902 option would specify a new queue, which would largely negate the previously stated advantage  
 29903 of using *qalter*; furthermore, the *qmove* utility provides a superior means of moving jobs.

29904 Each of the following paragraphs provides the rationale for a *qalter* option.

29905 Additional rationale concerning these options can be found in the rationale for the *qsub* utility.

29906 The **-a** option allows users to alter the date and time at which a batch job becomes eligible to  
 29907 run.

29908 The **-A** option allows users to change the account that will be charged for the resources  
 29909 consumed by the batch job. Support for the **-A** option is mandatory for conforming  
 29910 implementations of *qalter*, even though support of accounting is optional for servers. Whether or  
 29911 not to support accounting is left to the implementor of the server, but mandatory support of the  
 29912 **-A** option assures users of a consistent interface and allows them to control accounting on  
 29913 servers that support accounting.

29914 The **-c** option allows users to alter the checkpointing interval of a batch job. A checkpointing  
 29915 system, which is not defined by IEEE Std 1003.1-2001, allows recovery of a batch job at the most  
 29916 recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that consume  
 29917 expensive computing time or must meet a critical schedule. Users should be allowed to make  
 29918 the tradeoff between the overhead of checkpointing and the risk to the timely completion of the  
 29919 batch job; therefore, this volume of IEEE Std 1003.1-2001 provides the checkpointing interval  
 29920 option. Support for checkpointing is optional for servers.

29921 The **-e** option allows users to alter the name and location of the standard error stream written by  
 29922 a batch job. However, the path of the standard error stream is meaningless if the value of the  
 29923 *Join\_Path* attribute of the batch job is TRUE.

29924 The **-h** option allows users to set the hold type in the *Hold\_Types* attribute of a batch job. The  
 29925 *qhold* and *qrls* utilities add or remove hold types to the *Hold\_Types* attribute, respectively. The **-h**

- 29926 option has been modified to allow for implementation-defined hold types.
- 29927 The **-j** option allows users to alter the decision to join (merge) the standard error stream of the  
29928 batch job with the standard output stream of the batch job.
- 29929 The **-l** option allows users to change the resource limits imposed on a batch job.
- 29930 The **-m** option allows users to modify the list of points in the life of a batch job at which the  
29931 designated users will receive mail notification.
- 29932 The **-M** option allows users to alter the list of users who will receive notification about events in  
29933 the life of a batch job.
- 29934 The **-N** option allows users to change the name of a batch job.
- 29935 The **-o** option allows users to alter the name and path to which the standard output stream of  
29936 the batch job will be written.
- 29937 The **-P** option allows users to modify the priority of a batch job. Support for priority is optional  
29938 for batch servers.
- 29939 The **-r** option allows users to alter the rerunability status of a batch job.
- 29940 The **-S** option allows users to change the name and location of the shell image that will be  
29941 invoked to interpret the script of the batch job. This option has been modified to allow a list of  
29942 shell name and locations associated with different hosts.
- 29943 The **-u** option allows users to change the user identifier under which the batch job will execute.
- 29944 The *job\_identifier* operand syntax is provided so that the user can differentiate between the  
29945 originating and destination (or executing) batch server. These may or may not be the same. The  
29946 *.server\_name* portion identifies the originating batch server, while the *@server* portion identifies  
29947 the destination batch server.
- 29948 Historically, the *qalter* utility has been a component of the Network Queuing System (NQS), the  
29949 existing practice from which this utility has been derived.
- 29950 **FUTURE DIRECTIONS**
- 29951 None.
- 29952 **SEE ALSO**
- 29953 Chapter 3 (on page 101), *qdel*, *qhold*, *qmove*, *qrls*, *qsub*, *touch*
- 29954 **CHANGE HISTORY**
- 29955 Derived from IEEE Std 1003.2d-1994.
- 29956 **Issue 6**
- 29957 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.
- 29958 IEEE PASC Interpretation 1003.2 #182 is applied, clarifying the description of the **-a** option.

29959 **NAME**

29960 qdel — delete batch jobs

29961 **SYNOPSIS**29962 BE qdel *job\_identifier* ...

29963

29964 **DESCRIPTION**

29965 A batch job is deleted by sending a request to the batch server that manages the batch job. A  
 29966 batch job that has been deleted is no longer subject to management by batch services.

29967 The *qdel* utility is a user-accessible client of batch services that requests the deletion of one or  
 29968 more batch jobs.

29969 The *qdel* utility shall request a batch server to delete those batch jobs for which a batch  
 29970 *job\_identifier* is presented to the utility.

29971 The *qdel* utility shall delete batch jobs in the order in which their batch *job\_identifiers* are  
 29972 presented to the utility.

29973 If the *qdel* utility fails to process any batch *job\_identifier* successfully, the utility shall proceed to  
 29974 process the remaining batch *job\_identifiers*, if any.

29975 The *qdel* utility shall delete each batch job by sending a *Delete Job Request* to the batch server that  
 29976 manages the batch job.

29977 The *qdel* utility shall not exit until the batch job corresponding to each successfully processed  
 29978 batch *job\_identifier* has been deleted.

29979 **OPTIONS**

29980 None.

29981 **OPERANDS**

29982 The *qdel* utility shall accept one or more operands that conform to the syntax for a batch  
 29983 *job\_identifier* (see Section 3.3.1 (on page 122)).

29984 **STDIN**

29985 Not used.

29986 **INPUT FILES**

29987 None.

29988 **ENVIRONMENT VARIABLES**29989 The following environment variables shall affect the execution of *qdel*:

29990 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 29991 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 29992 Internationalization Variables for the precedence of internationalization variables  
 29993 used to determine the values of locale categories.)

29994 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 29995 internationalization variables.

29996 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 29997 characters (for example, single-byte as opposed to multi-byte characters in  
 29998 arguments).

29999 *LC\_MESSAGES*

30000 Determine the locale that should be used to affect the format and contents of  
 30001 diagnostic messages written to standard error.

30002            *LOGNAME*   Determine the login name of the user.

30003 **ASYNCHRONOUS EVENTS**

30004            Default.

30005 **STDOUT**

30006            An implementation of the *qdel* utility may write informative messages to standard output.

30007 **STDERR**

30008            The standard error shall be used only for diagnostic messages.

30009 **OUTPUT FILES**

30010            None.

30011 **EXTENDED DESCRIPTION**

30012            None.

30013 **EXIT STATUS**

30014            The following exit values shall be returned:

30015            0   Successful completion.

30016            >0   An error occurred.

30017 **CONSEQUENCES OF ERRORS**

30018            In addition to the default behavior, the *qdel* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job\_identifier* does not exist on the server. Whether or not the *qdel* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

30023 **APPLICATION USAGE**

30024            None.

30025 **EXAMPLES**

30026            None.

30027 **RATIONALE**

30028            The *qdel* utility allows users and administrators to delete jobs.

30029            The *qdel* utility provides functionality that is not otherwise available. For example, the *kill* utility of the operating system does not suffice. First, to use the *kill* utility, the user might have to log in on a remote node, because the *kill* utility does not operate across the network. Second, unlike *qdel*, *kill* cannot remove jobs from queues. Lastly, the arguments of the *qdel* utility are job identifiers rather than process identifiers, and so this utility can be passed the output of the *qselect* utility, thus providing users with a means of deleting a list of jobs.

30035            Because a set of jobs can be selected using the *qselect* utility, the *qdel* utility has not been complicated with options that provide for selection of jobs. Instead, the batch jobs to be deleted are identified individually by their job identifiers.

30038            Historically, the *qdel* utility has been a component of NQS, the existing practice on which it is based. However, the *qdel* utility defined in this volume of IEEE Std 1003.1-2001 does not provide an option for specifying a signal number to send to the batch job prior to the killing of the process; that capability has been subsumed by the *qsig* utility.

30042            A discussion was held about the delays of networking and the possibility that the batch server may never respond, due to a down router, down batch server, or other network mishap. The DESCRIPTION records this under the words “fails to process any job identifier”. In the broad sense, the network problem is also an error, which causes the failure to process the batch job

30046 identifier.

30047 **FUTURE DIRECTIONS**

30048 None.

30049 **SEE ALSO**

30050 Chapter 3 (on page 101), *kill*, *qselect*, *qsig*

30051 **CHANGE HISTORY**

30052 Derived from IEEE Std 1003.2d-1994.

30053 **Issue 6**

30054 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30055 **NAME**

30056 qhold — hold batch jobs

30057 **SYNOPSIS**30058 BE qhold [-h *hold\_list*] *job\_identifier* ...

30059

30060 **DESCRIPTION**

30061 A hold is placed on a batch job by a request to the batch server that manages the batch job. A  
 30062 batch job that has one or more holds is not eligible for execution. The *qhold* utility is a user-  
 30063 accessible client of batch services that requests one or more types of hold to be placed on one or  
 30064 more batch jobs.

30065 The *qhold* utility shall place holds on those batch jobs for which a batch *job\_identifier* is presented  
 30066 to the utility.

30067 The *qhold* utility shall place holds on batch jobs in the order in which their batch *job\_identifiers*  
 30068 are presented to the utility. If the *qhold* utility fails to process any batch *job\_identifier* successfully,  
 30069 the utility shall proceed to process the remaining batch *job\_identifiers*, if any.

30070 The *qhold* utility shall place holds on each batch job by sending a *Hold Job Request* to the batch  
 30071 server that manages the batch job.

30072 The *qhold* utility shall not exit until holds have been placed on the batch job corresponding to  
 30073 each successfully processed batch *job\_identifier*.

30074 **OPTIONS**

30075 The *qhold* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 30076 12.2, Utility Syntax Guidelines.

30077 The following option shall be supported by the implementation:

30078 **-h *hold\_list*** Define the types of holds to be placed on the batch job.

30079 The *qhold -h* option shall accept a value for the *hold\_list* option-argument that is a  
 30080 string of alphanumeric characters in the portable character set (see the Base  
 30081 Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set).

30082 The *qhold* utility shall accept a value for the *hold\_list* option-argument that is a  
 30083 string of one or more of the characters 'u', 's', or 'o', or the single character  
 30084 'n'.

30085 For each unique character in the *hold\_list* option-argument, the *qhold* utility shall  
 30086 add a value to the *Hold\_Types* attribute of the batch job as follows, each  
 30087 representing a different hold type:

30088 u USER

30089 s SYSTEM

30090 o OPERATOR

30091 If any of these characters are duplicated in the *hold\_list* option-argument, the  
 30092 duplicates shall be ignored.

30093 An existing *Hold\_Types* attribute can be cleared by the following hold type:

30094 n NO\_HOLD

30095 The *qhold* utility shall consider it an error if any hold type other than 'n' is  
 30096 combined with hold type 'n'.



30097 Strictly conforming applications shall not repeat any of the characters 'u', 's',  
 30098 'o', or 'n' within the *hold\_list* option-argument. The *qhold* utility shall permit the  
 30099 repetition of characters, but shall not assign additional meaning to the repeated  
 30100 characters.

30101 An implementation may define other hold types. The conformance document for  
 30102 an implementation shall describe any additional hold types, how they are  
 30103 specified, their internal behavior, and how they affect the behavior of the utility.

30104 If the **-h** option is not presented to the *qhold* utility, the implementation shall set  
 30105 the *Hold\_Types* attribute to USER.

#### 30106 **OPERANDS**

30107 The *qhold* utility shall accept one or more operands that conform to the syntax for a batch  
 30108 *job\_identifier* (see Section 3.3.1 (on page 122)).

#### 30109 **STDIN**

30110 Not used.

#### 30111 **INPUT FILES**

30112 None.

#### 30113 **ENVIRONMENT VARIABLES**

30114 The following environment variables shall affect the execution of *qhold*:

30115 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30116 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 30117 Internationalization Variables for the precedence of internationalization variables  
 30118 used to determine the values of locale categories.)

30119 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30120 internationalization variables.

30121 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30122 characters (for example, single-byte as opposed to multi-byte characters in  
 30123 arguments).

30124 *LC\_MESSAGES*

30125 Determine the locale that should be used to affect the format and contents of  
 30126 diagnostic messages written to standard error.

30127 *LOGNAME* Determine the login name of the user.

#### 30128 **ASYNCHRONOUS EVENTS**

30129 Default.

#### 30130 **STDOUT**

30131 None.

#### 30132 **STDERR**

30133 The standard error shall be used only for diagnostic messages.

#### 30134 **OUTPUT FILES**

30135 None.

#### 30136 **EXTENDED DESCRIPTION**

30137 None.

**30138 EXIT STATUS**

30139 The following exit values shall be returned:

30140 0 Successful completion.

30141 >0 An error occurred.

**30142 CONSEQUENCES OF ERRORS**

30143 In addition to the default behavior, the *qhold* utility shall not be required to write a diagnostic  
30144 message to standard error when the error reply received from a batch server indicates that the  
30145 batch *job\_identifier* does not exist on the server. Whether or not the *qhold* utility waits to output  
30146 the diagnostic message while attempting to locate the job on other servers is implementation-  
30147 defined.

**30148 APPLICATION USAGE**

30149 None.

**30150 EXAMPLES**

30151 None.

**30152 RATIONALE**

30153 The *qhold* utility allows users to place a hold on one or more jobs. A hold makes a batch job  
30154 ineligible for execution.

30155 The *qhold* utility has options that allow the user to specify the type of hold. Should the user wish  
30156 to place a hold on a set of jobs that meet a selection criteria, such a list of jobs can be acquired  
30157 using the *qselect* utility.

30158 The **-h** option allows the user to specify the type of hold that is to be placed on the job. This  
30159 option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The  
30160 USER and OPERATOR holds are distinct. The batch server that manages the batch job will verify  
30161 that the user is authorized to set the specified hold for the batch job.

30162 Mail is not required on hold because the administrator has the tools and libraries to build this  
30163 option if he or she wishes.

30164 Historically, the *qhold* utility has been a part of some existing batch systems, although it has not  
30165 traditionally been a part of the NQS.

**30166 FUTURE DIRECTIONS**

30167 None.

**30168 SEE ALSO**

30169 Chapter 3 (on page 101), *qselect*

**30170 CHANGE HISTORY**

30171 Derived from IEEE Std 1003.2d-1994.

**30172 Issue 6**

30173 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30174 **NAME**

30175           qmove — move batch jobs

30176 **SYNOPSIS**30177 BE       *qmove destination job\_identifier ...*

30178

30179 **DESCRIPTION**

30180       To move a batch job is to remove the batch job from the batch queue in which it resides and  
 30181       instantiate the batch job in another batch queue. A batch job is moved by a request to the batch  
 30182       server that manages the batch job. The *qmove* utility is a user-accessible batch client that requests  
 30183       the movement of one or more batch jobs.

30184       The *qmove* utility shall move those batch jobs, and only those batch jobs, for which a batch  
 30185       *job\_identifier* is presented to the utility.

30186       The *qmove* utility shall move batch jobs in the order in which the corresponding batch  
 30187       *job\_identifiers* are presented to the utility.

30188       If the *qmove* utility fails to process a batch *job\_identifier* successfully, the utility shall proceed to  
 30189       process the remaining batch *job\_identifiers*, if any.

30190       The *qmove* utility shall move batch jobs by sending a *Move Job Request* to the batch server that  
 30191       manages each batch job. The *qmove* utility shall not exit before the batch jobs corresponding to all  
 30192       successfully processed batch *job\_identifiers* have been moved.

30193 **OPTIONS**

30194           None.

30195 **OPERANDS**

30196       The *qmove* utility shall accept one operand that conforms to the syntax for a destination (see  
 30197       Section 3.3.2 (on page 123)).

30198       The *qmove* utility shall accept one or more operands that conform to the syntax for a batch  
 30199       *job\_identifier* (see Section 3.3.1 (on page 122)).

30200 **STDIN**

30201           Not used.

30202 **INPUT FILES**

30203           None.

30204 **ENVIRONMENT VARIABLES**30205       The following environment variables shall affect the execution of *qmove*:

30206       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 30207       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 30208       Internationalization Variables for the precedence of internationalization variables  
 30209       used to determine the values of locale categories.)

30210       *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 30211       internationalization variables.

30212       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 30213       characters (for example, single-byte as opposed to multi-byte characters in  
 30214       arguments).

30215       *LC\_MESSAGES*

30216       Determine the locale that should be used to affect the format and contents of  
 30217       diagnostic messages written to standard error.

30218 *LOGNAME* Determine the login name of the user.

30219 **ASYNCHRONOUS EVENTS**

30220 Default.

30221 **STDOUT**

30222 None.

30223 **STDERR**

30224 The standard error shall be used only for diagnostic messages.

30225 **OUTPUT FILES**

30226 None.

30227 **EXTENDED DESCRIPTION**

30228 None.

30229 **EXIT STATUS**

30230 The following exit values shall be returned:

30231 0 Successful completion.

30232 >0 An error occurred.

30233 **CONSEQUENCES OF ERRORS**

30234 In addition to the default behavior, the *qmove* utility shall not be required to write a diagnostic  
30235 message to standard error when the error reply received from a batch server indicates that the  
30236 batch *job\_identifier* does not exist on the server. Whether or not the *qmove* utility waits to output  
30237 the diagnostic message while attempting to locate the job on other servers is implementation-  
30238 defined.

30239 **APPLICATION USAGE**

30240 None.

30241 **EXAMPLES**

30242 None.

30243 **RATIONALE**

30244 The *qmove* utility allows users to move jobs between queues.

30245 The alternative to using the *qmove* utility—deleting the batch job and requeuing it—entails  
30246 considerably more typing.

30247 Since the means of selecting jobs based on attributes has been encapsulated in the *qselect* utility,  
30248 the only option of the *qmove* utility concerns authorization. The **-u** option provides the user with  
30249 the convenience of changing the user identifier under which the batch job will execute.  
30250 Minimalism and consistency have taken precedence over convenience; the **-u** option has been  
30251 deleted because the equivalent capability exists with the **-u** option of the *qalter* utility.

30252 **FUTURE DIRECTIONS**

30253 None.

30254 **SEE ALSO**

30255 Chapter 3 (on page 101), *qalter*, *qselect*

30256 **CHANGE HISTORY**

30257 Derived from IEEE Std 1003.2d-1994.

30258 **Issue 6**

30259 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30260 **NAME**

30261 qmsg — send message to batch jobs

30262 **SYNOPSIS**30263 BE qmsg [-E] [-O] *message\_string* *job\_identifier* ...

30264

30265 **DESCRIPTION**

30266 To send a message to a batch job is to request that a server write a message string into one or  
 30267 more output files of the batch job. A message is sent to a batch job by a request to the batch  
 30268 server that manages the batch job. The *qmsg* utility is a user-accessible batch client that requests  
 30269 the sending of messages to one or more batch jobs.

30270 The *qmsg* utility shall write messages into the files of batch jobs by sending a *Job Message Request*  
 30271 to the batch server that manages the batch job. The *qmsg* utility shall not directly write the  
 30272 message into the files of the batch job.

30273 The *qmsg* utility shall send a *Job Message Request* for those batch jobs, and only those batch jobs,  
 30274 for which a batch *job\_identifier* is presented to the utility.

30275 The *qmsg* utility shall send *Job Message Requests* for batch jobs in the order in which their batch  
 30276 *job\_identifiers* are presented to the utility.

30277 If the *qmsg* utility fails to process any batch *job\_identifier* successfully, the utility shall proceed to  
 30278 process the remaining batch *job\_identifiers*, if any.

30279 The *qmsg* utility shall not exit before a *Job Message Request* has been sent to the server that  
 30280 manages the batch job that corresponds to each successfully processed batch *job\_identifier*.

30281 **OPTIONS**

30282 The *qmsg* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 30283 12.2, Utility Syntax Guidelines.

30284 The following options shall be supported by the implementation:

30285 **-E** Specify that the message is written to the standard error of each batch job.

30286 The *qmsg* utility shall write the message into the standard error of the batch job.

30287 **-O** Specify that the message is written to the standard output of each batch job.

30288 The *qmsg* utility shall write the message into the standard output of the batch job.

30289 If neither the **-O** nor the **-E** option is presented to the *qmsg* utility, the utility shall write the  
 30290 message into an implementation-defined file. The conformance document for the  
 30291 implementation shall describe the name and location of the implementation-defined file. If both  
 30292 the **-O** and the **-E** options are presented to the *qmsg* utility, then the utility shall write the  
 30293 messages to both standard output and standard error.

30294 **OPERANDS**

30295 The *qmsg* utility shall accept a minimum of two operands, *message\_string* and one or more batch  
 30296 *job\_identifiers*.

30297 The *message\_string* operand shall be the string to be written to one or more output files of the  
 30298 batch job followed by a <newline>. If the string contains <blank>s, then the application shall  
 30299 ensure that the string is quoted. The *message\_string* shall be encoded in the portable character set  
 30300 (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set).

30301 All remaining operands are batch *job\_identifiers* that conform to the syntax for a batch  
 30302 *job\_identifier* (see Section 3.3.1 (on page 122)).

30303 **STDIN**

30304 Not used.

30305 **INPUT FILES**

30306 None.

30307 **ENVIRONMENT VARIABLES**30308 The following environment variables shall affect the execution of *qmsg*:

30309 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30310 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 30311 Internationalization Variables for the precedence of internationalization variables  
 30312 used to determine the values of locale categories.)

30313 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30314 internationalization variables.

30315 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30316 characters (for example, single-byte as opposed to multi-byte characters in  
 30317 arguments).

30318 *LC\_MESSAGES*

30319 Determine the locale that should be used to affect the format and contents of  
 30320 diagnostic messages written to standard error.

30321 *LOGNAME* Determine the login name of the user.30322 **ASYNCHRONOUS EVENTS**

30323 Default.

30324 **STDOUT**

30325 None.

30326 **STDERR**

30327 The standard error shall be used only for diagnostic messages.

30328 **OUTPUT FILES**

30329 None.

30330 **EXTENDED DESCRIPTION**

30331 None.

30332 **EXIT STATUS**

30333 The following exit values shall be returned:

30334 0 Successful completion.

30335 &gt;0 An error occurred.

30336 **CONSEQUENCES OF ERRORS**

30337 In addition to the default behavior, the *qmsg* utility shall not be required to write a diagnostic  
 30338 message to standard error when the error reply received from a batch server indicates that the  
 30339 batch *job\_identifier* does not exist on the server. Whether or not the *qmsg* utility waits to output  
 30340 the diagnostic message while attempting to locate the job on other servers is implementation-  
 30341 defined.

**30342 APPLICATION USAGE**

30343 None.

**30344 EXAMPLES**

30345 None.

**30346 RATIONALE**

30347 The *qmsg* utility allows users to write messages into the output files of running jobs. Users,  
30348 including operators and administrators, have a number of occasions when they want to place  
30349 messages in the output files of a batch job. For example, if a disk that is being used by a batch job  
30350 is showing errors, the operator might note this in the standard error stream of the batch job.

30351 The options of the *qmsg* utility provide users with the means of placing the message in the  
30352 output stream of their choice. The default output stream for the message—if the user does not  
30353 designate an output stream—is implementation-defined, since many implementations will  
30354 provide, as an extension to this volume of IEEE Std 1003.1-2001, a log file that shows the history  
30355 of utility execution.

30356 If users wish to send a message to a set of jobs that meet a selection criteria, the *qselect* utility can  
30357 be used to acquire the appropriate list of job identifiers.

30358 The **-E** option allows users to place the message in the standard error stream of the batch job.

30359 The **-O** option allows users to place the message in the standard output stream of the batch job.

30360 Historically, the *qmsg* utility is an existing practice in the offerings of one or more implementors  
30361 of an NQS-derived batch system. The utility has been found to be useful enough that it deserves  
30362 to be included in this volume of IEEE Std 1003.1-2001.

**30363 FUTURE DIRECTIONS**

30364 None.

**30365 SEE ALSO**

30366 Chapter 3 (on page 101), *qselect*

**30367 CHANGE HISTORY**

30368 Derived from IEEE Std 1003.2d-1994.

**30369 Issue 6**

30370 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.



30371 **NAME**

30372 qrerun — rerun batch jobs

30373 **SYNOPSIS**30374 BE qrerun *job\_identifier* ...

30375

30376 **DESCRIPTION**

30377 To rerun a batch job is to terminate the session leader of the batch job, delete any associated  
 30378 checkpoint files, and return the batch job to the batch queued state. A batch job is rerun by a  
 30379 request to the batch server that manages the batch job. The *qrerun* utility is a user-accessible  
 30380 batch client that requests the rerunning of one or more batch jobs.

30381 The *qrerun* utility shall rerun those batch jobs for which a batch *job\_identifier* is presented to the  
 30382 utility.

30383 The *qrerun* utility shall rerun batch jobs in the order in which their batch *job\_identifiers* are  
 30384 presented to the utility.

30385 If the *qrerun* utility fails to process any batch *job\_identifier* successfully, the utility shall proceed  
 30386 to process the remaining batch *job\_identifiers*, if any.

30387 The *qrerun* utility shall rerun batch jobs by sending a *Rerun Job Request* to the batch server that  
 30388 manages each batch job.

30389 For each successfully processed batch *job\_identifier*, the *qrerun* utility shall have rerun the  
 30390 corresponding batch job at the time the utility exits.

30391 **OPTIONS**

30392 None.

30393 **OPERANDS**

30394 The *qrerun* utility shall accept one or more operands that conform to the syntax for a batch  
 30395 *job\_identifier* (see Section 3.3.1 (on page 122)).

30396 **STDIN**

30397 Not used.

30398 **INPUT FILES**

30399 None.

30400 **ENVIRONMENT VARIABLES**30401 The following environment variables shall affect the execution of *qrerun*:

30402 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30403 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 30404 Internationalization Variables for the precedence of internationalization variables  
 30405 used to determine the values of locale categories.)

30406 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30407 internationalization variables.

30408 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30409 characters (for example, single-byte as opposed to multi-byte characters in  
 30410 arguments).

30411 *LC\_MESSAGES*

30412 Determine the locale that should be used to affect the format and contents of  
 30413 diagnostic messages written to standard error.

30414 *LOGNAME* Determine the login name of the user.

30415 **ASYNCHRONOUS EVENTS**

30416 Default.

30417 **STDOUT**

30418 None.

30419 **STDERR**

30420 The standard error shall be used only for diagnostic messages.

30421 **OUTPUT FILES**

30422 None.

30423 **EXTENDED DESCRIPTION**

30424 None.

30425 **EXIT STATUS**

30426 The following exit values shall be returned:

30427 0 Successful completion.

30428 >0 An error occurred.

30429 **CONSEQUENCES OF ERRORS**

30430 In addition to the default behavior, the *qrerun* utility shall not be required to write a diagnostic  
30431 message to standard error when the error reply received from a batch server indicates that the  
30432 batch *job\_identifier* does not exist on the server. Whether or not the *qrerun* utility waits to output  
30433 the diagnostic message while attempting to locate the job on other servers is implementation-  
30434 defined.

30435 **APPLICATION USAGE**

30436 None.

30437 **EXAMPLES**

30438 None.

30439 **RATIONALE**

30440 The *qrerun* utility allows users to cause jobs in the running state to exit and rerun.

30441 The *qrerun* utility is a new utility, *vis-a-vis* existing practice, that has been defined in this volume  
30442 of IEEE Std 1003.1-2001 to correct user-perceived deficiencies in the existing practice.

30443 **FUTURE DIRECTIONS**

30444 None.

30445 **SEE ALSO**

30446 Chapter 3 (on page 101)

30447 **CHANGE HISTORY**

30448 Derived from IEEE Std 1003.2d-1994.

30449 **Issue 6**

30450 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30451 **NAME**

30452 qrln — release batch jobs

30453 **SYNOPSIS**30454 BE qrln [-h *hold\_list*] *job\_identifier* ...

30455

30456 **DESCRIPTION**

30457 A batch job might have one or more holds, which prevent the batch job from executing. A batch  
 30458 job from which all the holds have been removed becomes eligible for execution and is said to  
 30459 have been released. A batch job hold is removed by sending a request to the batch server that  
 30460 manages the batch job. The *qrln* utility is a user-accessible client of batch services that requests  
 30461 holds be removed from one or more batch jobs.

30462 The *qrln* utility shall remove one or more holds from those batch jobs for which a batch  
 30463 *job\_identifier* is presented to the utility.

30464 The *qrln* utility shall remove holds from batch jobs in the order in which their batch *job\_identifiers*  
 30465 are presented to the utility.

30466 If the *qrln* utility fails to process a batch *job\_identifier* successfully, the utility shall proceed to  
 30467 process the remaining batch *job\_identifiers*, if any.

30468 The *qrln* utility shall remove holds on each batch job by sending a *Release Job Request* to the batch  
 30469 server that manages the batch job.

30470 The *qrln* utility shall not exit until the holds have been removed from the batch job  
 30471 corresponding to each successfully processed batch *job\_identifier*.

30472 **OPTIONS**

30473 The *qrln* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 30474 12.2, Utility Syntax Guidelines.

30475 The following option shall be supported by the implementation:

30476 **-h *hold\_list*** Define the types of holds to be removed from the batch job.

30477 The *qrln* **-h** option shall accept a value for the *hold\_list* option-argument that is a  
 30478 string of alphanumeric characters in the portable character set (see the Base  
 30479 Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set).

30480 The *qrln* utility shall accept a value for the *hold\_list* option-argument that is a string  
 30481 of one or more of the characters 'u', 's', or 'o', or the single character 'n'.

30482 For each unique character in the *hold\_list* option-argument, the *qrln* utility shall add  
 30483 a value to the *Hold\_Types* attribute of the batch job as follows, each representing a  
 30484 different hold type:

30485 u USER

30486 s SYSTEM

30487 o OPERATOR

30488 If any of these characters are duplicated in the *hold\_list* option-argument, the  
 30489 duplicates shall be ignored.

30490 An existing *Hold\_Types* attribute can be cleared by the following hold type:

30491 n NO\_HOLD

30492 The *qrls* utility shall consider it an error if any hold type other than 'n' is  
 30493 combined with hold type 'n'.

30494 Strictly conforming applications shall not repeat any of the characters 'u', 's',  
 30495 'o', or 'n' within the *hold\_list* option-argument. The *qrls* utility shall permit the  
 30496 repetition of characters, but shall not assign additional meaning to the repeated  
 30497 characters.

30498 An implementation may define other hold types. The conformance document for  
 30499 an implementation shall describe any additional hold types, how they are  
 30500 specified, their internal behavior, and how they affect the behavior of the utility.

30501 If the **-h** option is not presented to the *qrls* utility, the implementation shall remove  
 30502 the USER hold in the *Hold\_Types* attribute.

#### 30503 OPERANDS

30504 The *qrls* utility shall accept one or more operands that conform to the syntax for a batch  
 30505 *job\_identifier* (see Section 3.3.1 (on page 122)).

#### 30506 STDIN

30507 Not used.

#### 30508 INPUT FILES

30509 None.

#### 30510 ENVIRONMENT VARIABLES

30511 The following environment variables shall affect the execution of *qrls*:

30512 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30513 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 30514 Internationalization Variables for the precedence of internationalization variables  
 30515 used to determine the values of locale categories.)

30516 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30517 internationalization variables.

30518 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30519 characters (for example, single-byte as opposed to multi-byte characters in  
 30520 arguments).

30521 *LC\_MESSAGES*

30522 Determine the locale that should be used to affect the format and contents of  
 30523 diagnostic messages written to standard error.

30524 *LOGNAME* Determine the login name of the user.

#### 30525 ASYNCHRONOUS EVENTS

30526 Default.

#### 30527 STDOUT

30528 None.

#### 30529 STDERR

30530 The standard error shall be used only for diagnostic messages.

#### 30531 OUTPUT FILES

30532 None.

**30533 EXTENDED DESCRIPTION**

30534 None.

**30535 EXIT STATUS**

30536 The following exit values shall be returned:

30537 0 Successful completion.

30538 >0 An error occurred.

**30539 CONSEQUENCES OF ERRORS**

30540 In addition to the default behavior, the *qrIs* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job\_identifier* does not exist on the server. Whether or not the *qrIs* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

**30545 APPLICATION USAGE**

30546 None.

**30547 EXAMPLES**

30548 None.

**30549 RATIONALE**

30550 The *qrIs* utility allows users, operators, and administrators to remove holds from jobs.

30551 The *qrIs* utility does not support any job selection options or wildcard arguments. Users may acquire a list of jobs selected by attributes using the *qselect* utility. For example, a user could select all of their held jobs.

30552 The **-h** option allows the user to specify the type of hold that is to be removed. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The batch server that manages the batch job will verify whether the user is authorized to remove the specified hold for the batch job. If more than one type of hold has been placed on the batch job, a user may wish to remove only some of them.

30553 Mail is not required on release because the administrator has the tools and libraries to build this option if required.

30554 The *qrIs* utility is a new utility *vis-a-vis* existing practice; it has been defined in this volume of IEEE Std 1003.1-2001 as the natural complement to the *qhold* utility.

**30563 FUTURE DIRECTIONS**

30564 None.

**30565 SEE ALSO**

30566 Chapter 3 (on page 101), *qhold*, *qselect*

**30567 CHANGE HISTORY**

30568 Derived from IEEE Std 1003.2d-1994.

**30569 Issue 6**

30570 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30571 **NAME**

30572 qselect — select batch jobs

30573 **SYNOPSIS**

```
30574 BE qselect [-a [op]date_time] [-A account_string] [-c [op]interval]
30575 [-h hold_list] [-l resource_list] [-N name] [-p [op]priority]
30576 [-q destination] [-r y|n] [-s states] [-u user_list]
30577
```

30578 **DESCRIPTION**

30579 To select a set of batch jobs is to return the batch *job\_identifiers* for each batch job that meets a list  
 30580 of selection criteria. A set of batch jobs is selected by a request to a batch server. The *qselect*  
 30581 utility is a user-accessible batch client that requests the selection of batch jobs.

30582 Upon successful completion, the *qselect* utility shall have returned a list of zero or more batch  
 30583 *job\_identifiers* that meet the criteria specified by the options and option-arguments presented to  
 30584 the utility.

30585 The *qselect* utility shall select batch jobs by sending a *Select Jobs Request* to a batch server. The  
 30586 *qselect* utility shall not exit until the server replies to each request generated.

30587 For each option presented to the *qselect* utility, the utility shall restrict the set of selected batch  
 30588 jobs as described in the OPTIONS section.

30589 The *qselect* utility shall not restrict selection of batch jobs except by authorization and as required  
 30590 by the options presented to the utility.

30591 When an option is specified with a mandatory or optional *op* component to the option-  
 30592 argument, then *op* shall specify a relation between the value of a certain batch job attribute and  
 30593 the *value* component of the option-argument. If an *op* is allowable on an option, then the  
 30594 description of the option letter indicates the *op* as either mandatory or optional. Acceptable  
 30595 strings for the *op* component, and the relation the string indicates, are shown in the following  
 30596 list:

- |                |      |                                                                                                                                   |
|----------------|------|-----------------------------------------------------------------------------------------------------------------------------------|
| 30597<br>30598 | .eq. | The value represented by the attribute of the batch job is equal to the value represented by the option-argument.                 |
| 30599<br>30600 | .ge. | The value represented by the attribute of the batch job is greater than or equal to the value represented by the option-argument. |
| 30601<br>30602 | .gt. | The value represented by the attribute of the batch job is greater than the value represented by the option-argument.             |
| 30603<br>30604 | .lt. | The value represented by the attribute of the batch job is less than the value represented by the option-argument.                |
| 30605<br>30606 | .le. | The value represented by the attribute of the batch job is less than or equal to the value represented by the option-argument.    |
| 30607<br>30608 | .ne. | The value represented by the attribute of the batch job is not equal to the value represented by the option-argument.             |

30609 **OPTIONS**

30610 The *qselect* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 30611 12.2, Utility Syntax Guidelines.

30612 The following options shall be supported by the implementation:

- |                |                  |                                                             |
|----------------|------------------|-------------------------------------------------------------|
| 30613<br>30614 | -a [op]date_time | Restrict selection to a specific time, or a range of times. |
|----------------|------------------|-------------------------------------------------------------|

30615 The *qselect* utility shall select only batch jobs for which the value of the  
 30616 *Execution\_Time* attribute is related to the Epoch equivalent of the local time  
 30617 expressed by the value of the *date\_time* component of the option-argument in the  
 30618 manner indicated by the value of the *op* component of the option-argument.

30619 The *qselect* utility shall accept a *date\_time* component of the option-argument that  
 30620 conforms to the syntax of the *time* operand of the *touch* utility.

30621 If the *op* component of the option-argument is not presented to the *qselect* utility,  
 30622 the utility shall select batch jobs for which the *Execution\_Time* attribute is equal to  
 30623 the *date\_time* component of the option-argument.

30624 When comparing times, the *qselect* utility shall use the following definitions for the  
 30625 *op* component of the option-argument:

30626 .eq. The time represented by value of the *Execution\_Time* attribute of the batch  
 30627 job is equal to the time represented by the *date\_time* component of the  
 30628 option-argument.

30629 .ge. The time represented by value of the *Execution\_Time* attribute of the batch  
 30630 job is after or equal to the time represented by the *date\_time* component of  
 30631 the option-argument.

30632 .gt. The time represented by value of the *Execution\_Time* attribute of the batch  
 30633 job is after the time represented by the *date\_time* component of the  
 30634 option-argument.

30635 .lt. The time represented by value of the *Execution\_Time* attribute of the batch  
 30636 job is before the time represented by the *date\_time* component of the  
 30637 option-argument.

30638 .le. The time represented by value of the *Execution\_Time* attribute of the batch  
 30639 job is before or equal to the time represented by the *date\_time* component  
 30640 of the option-argument.

30641 .ne. The time represented by value of the *Execution\_Time* attribute of the batch  
 30642 job is not equal to the time represented by the *date\_time* component of the  
 30643 option-argument.

30644 The *qselect* utility shall accept the defined character strings for the *op* component of  
 30645 the option-argument.

30646 **-A** *account\_string*  
 30647 Restrict selection to the batch jobs charging a specified account.

30648 The *qselect* utility shall select only batch jobs for which the value of the  
 30649 *Account\_Name* attribute of the batch job matches the value of the *account\_string*  
 30650 option-argument.

30651 The syntax of the *account\_string* option-argument is unspecified.

30652 **-c** [*op*]*interval*  
 30653 Restrict selection to batch jobs within a range of checkpoint intervals.

30654 The *qselect* utility shall select only batch jobs for which the value of the *Checkpoint*  
 30655 attribute relates to the value of the *interval* component of the option-argument in  
 30656 the manner indicated by the value of the *op* component of the option-argument.

30657 If the *op* component of the option-argument is omitted, the *qselect* utility shall  
 30658 select batch jobs for which the value of the *Checkpoint* attribute is equal to the value

30659 of the *interval* component of the option-argument.

30660 When comparing checkpoint intervals, the *qselect* utility shall use the following

30661 definitions for the *op* component of the option-argument:

30662 .eq. The value of the *Checkpoint* attribute of the batch job equals the value of

30663 the *interval* component of the option-argument.

30664 .ge. The value of the *Checkpoint* attribute of the batch job is greater than or

30665 equal to the value of the *interval* component option-argument.

30666 .gt. The value of the *Checkpoint* attribute of the batch job is greater than the

30667 value of the *interval* component option-argument.

30668 .lt. The value of the *Checkpoint* attribute of the batch job is less than the value

30669 of the *interval* component option-argument.

30670 .le. The value of the *Checkpoint* attribute of the batch job is less than or equal

30671 to the value of the *interval* component option-argument.

30672 .ne. The value of the *Checkpoint* attribute of the batch job does not equal the

30673 value of the *interval* component option-argument.

30674 The *qselect* utility shall accept the defined character strings for the *op* component of

30675 the option-argument.

30676 The ordering relationship for the values of the interval option-argument is defined

30677 to be:

30678 'n' .gt. 's' .gt. 'c=minutes' .ge. 'c'

30679 When comparing *Checkpoint* attributes with an interval having the value of the

30680 single character 'u', only equality or inequality are valid comparisons.

30681 **-h hold\_list** Restrict selection to batch jobs that have a specific type of hold.

30682 The *qselect* utility shall select only batch jobs for which the value of the *Hold\_Types*

30683 attribute matches the value of the *hold\_list* option-argument.

30684 The *qselect* **-h** option shall accept a value for the *hold\_list* option-argument that is a

30685 string of alphanumeric characters in the portable character set (see the Base

30686 Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set).

30687 The *qselect* utility shall accept a value for the *hold\_list* option-argument that is a

30688 string of one or more of the characters 'u', 's', or 'o', or the single character

30689 'n'.

30690 Each unique character in the *hold\_list* option-argument of the *qselect* utility is

30691 defined as follows, each representing a different hold type:

30692 u USER

30693 s SYSTEM

30694 o OPERATOR

30695 If any of these characters are duplicated in the *hold\_list* option-argument, the

30696 duplicates shall be ignored.

30697 The *qselect* utility shall consider it an error if any hold type other than 'n' is

30698 combined with hold type 'n'.



30699 Strictly conforming applications shall not repeat any of the characters 'u', 's',  
 30700 'o', or 'n' within the *hold\_list* option-argument. The *qselect* utility shall permit  
 30701 the repetition of characters, but shall not assign additional meaning to the repeated  
 30702 characters.

30703 An implementation may define other hold types. The conformance document for  
 30704 an implementation shall describe any additional hold types, how they are  
 30705 specified, their internal behavior, and how they affect the behavior of the utility.

30706 **-I *resource\_list***  
 30707 Restrict selection to batch jobs with specified resource limits and attributes.

30708 The *qselect* utility shall accept a *resource\_list* option-argument with the following  
 30709 syntax:

30710 *resource\_name op value [, , resource\_name op value, , ...]*

30711 When comparing resource values, the *qselect* utility shall use the following  
 30712 definitions for the *op* component of the option-argument:

30713 .eq. The value of the resource of the same name in the *Resource\_List* attribute  
 30714 of the batch job equals the value of the value component of the option-  
 30715 argument.

30716 .ge. The value of the resource of the same name in the *Resource\_List* attribute  
 30717 of the batch job is greater than or equal to the value of the value  
 30718 component of the option-argument.

30719 .gt. The value of the resource of the same name in the *Resource\_List* attribute  
 30720 of the batch job is greater than the value of the value component of the  
 30721 option-argument.

30722 .lt. The value of the resource of the same name in the *Resource\_List* attribute  
 30723 of the batch job is less than the value of the value component of the  
 30724 option-argument.

30725 .ne. The value of the resource of the same name in the *Resource\_List* attribute  
 30726 of the batch job does not equal the value of the value component of the  
 30727 option-argument.

30728 .le. The value of the resource of the same name in the *Resource\_List* attribute  
 30729 of the batch job is less than or equal to the value of the value component  
 30730 of the option-argument.

30731 When comparing the limit of a *Resource\_List* attribute with the *value* component of  
 30732 the option-argument, if the limit, the value, or both are non-numeric, only equality  
 30733 or inequality are valid comparisons.

30734 The *qselect* utility shall select only batch jobs for which the values of the  
 30735 *resource\_names* listed in the *resource\_list* option-argument match the corresponding  
 30736 limits of the *Resource\_List* attribute of the batch job.

30737 Limits of *resource\_names* present in the *Resource\_List* attribute of the batch job that  
 30738 have no corresponding values in the *resource\_list* option-argument shall not be  
 30739 considered when selecting batch jobs.

30740 **-N *name*** Restrict selection to batch jobs with a specified name.

30741 The *qselect* utility shall select only batch jobs for which the value of the *Job\_Name*  
 30742 attribute matches the value of the *name* option-argument. The string specified in

30743 the *name* option-argument shall be passed, uninterpreted, to the server. This allows  
 30744 an implementation to match “wildcard” patterns against batch job names.

30745 An implementation shall describe in the conformance document the format it  
 30746 supports for matching against the *Job\_Name* attribute.

30747 **-p [op]priority**

30748 Restrict selection to batch jobs of the specified priority or range of priorities.

30749 The *qselect* utility shall select only batch jobs for which the value of the *Priority*  
 30750 attribute of the batch job relates to the value of the *priority* component of the  
 30751 option-argument in the manner indicated by the value of the *op* component of the  
 30752 option-argument.

30753 If the *op* component of the option-argument is omitted, the *qselect* utility shall  
 30754 select batch jobs for which the value of the *Priority* attribute of the batch job is  
 30755 equal to the value of the *priority* component of the option-argument.

30756 When comparing priority values, the *qselect* utility shall use the following  
 30757 definitions for the *op* component of the option-argument:

30758 .eq. The value of the *Priority* attribute of the batch job equals the value of the  
 30759 *priority* component of the option-argument.

30760 .ge. The value of the *Priority* attribute of the batch job is greater than or equal  
 30761 to the value of the *priority* component option-argument.

30762 .gt. The value of the *Priority* attribute of the batch job is greater than the value  
 30763 of the *priority* component option-argument.

30764 .lt. The value of the *Priority* attribute of the batch job is less than the value of  
 30765 the *priority* component option-argument.

30766 .lte. The value of the *Priority* attribute of the batch job is less than or equal to  
 30767 the value of the *priority* component option-argument.

30768 .ne. The value of the *Priority* attribute of the batch job does not equal the value  
 30769 of the *priority* component option-argument.

30770 **-q destination**

30771 Restrict selection to the specified batch queue or server, or both.

30772 The *qselect* utility shall select only batch jobs that are located at the destination  
 30773 indicated by the value of the *destination* option-argument.

30774 The destination defines a batch queue, a server, or a batch queue at a server.

30775 The *qselect* utility shall accept an option-argument for the **-q** option that conforms  
 30776 to the syntax for a destination. If the **-q** option is not presented to the *qselect* utility,  
 30777 the utility shall select batch jobs from all batch queues at the default batch server.

30778 If the option-argument describes only a batch queue, the *qselect* utility shall select  
 30779 only batch jobs from the batch queue of the specified name at the default batch  
 30780 server. The means by which *qselect* determines the default server is  
 30781 implementation-defined.

30782 If the option-argument describes only a batch server, the *qselect* utility shall select  
 30783 batch jobs from all the batch queues at that batch server.

30784 If the option-argument describes both a batch queue and a batch server, the *qselect*  
 30785 utility shall select only batch jobs from the specified batch queue at the specified

|       |                     |                                                                                                        |
|-------|---------------------|--------------------------------------------------------------------------------------------------------|
| 30786 |                     | server.                                                                                                |
| 30787 | <b>-r y n</b>       | Restrict selection to batch jobs with the specified rerunability status.                               |
| 30788 |                     | The <i>qselect</i> utility shall select only batch jobs for which the value of the <i>Rerunable</i>    |
| 30789 |                     | attribute of the batch job matches the value of the option-argument.                                   |
| 30790 |                     | The <i>qselect</i> utility shall accept a value for the option-argument that consists of               |
| 30791 |                     | either the single character 'y' or the single character 'n'. The character 'y'                         |
| 30792 |                     | represents the value TRUE, and the character 'n' represents the value FALSE.                           |
| 30793 | <b>-s states</b>    | Restrict selection to batch jobs in the specified states.                                              |
| 30794 |                     | The <i>qselect</i> utility shall accept an option-argument that consists of any combination            |
| 30795 |                     | of the characters 'e', 'q', 'r', 'w', 'h', and 't'.                                                    |
| 30796 |                     | Conforming applications shall not repeat any character in the option-argument.                         |
| 30797 |                     | The <i>qselect</i> utility shall permit the repetition of characters in the option-argument,           |
| 30798 |                     | but shall not assign additional meaning to repeated characters.                                        |
| 30799 |                     | The <i>qselect</i> utility shall interpret the characters in the <i>states</i> option-argument as      |
| 30800 |                     | follows:                                                                                               |
| 30801 |                     | e Represents the EXITING state.                                                                        |
| 30802 |                     | q Represents the QUEUED state.                                                                         |
| 30803 |                     | r Represents the RUNNING state.                                                                        |
| 30804 |                     | t Represents the TRANSITING state.                                                                     |
| 30805 |                     | h Represents the HELD state.                                                                           |
| 30806 |                     | w Represents the WAITING state.                                                                        |
| 30807 |                     | For each character in the <i>states</i> option-argument, the <i>qselect</i> utility shall select batch |
| 30808 |                     | jobs in the corresponding state.                                                                       |
| 30809 | <b>-u user_list</b> | Restrict selection to batch jobs owned by the specified user names.                                    |
| 30810 |                     | The <i>qselect</i> utility shall select only the batch jobs of those users specified in the            |
| 30811 |                     | <i>user_list</i> option-argument.                                                                      |
| 30812 |                     | The <i>qselect</i> utility shall accept a <i>user_list</i> option-argument that conforms to the        |
| 30813 |                     | following syntax:                                                                                      |
| 30814 |                     | <i>username</i> [@ <i>host</i> ] [, , <i>username</i> [@ <i>host</i> ] , , . . .]                      |
| 30815 |                     | The <i>qselect</i> utility shall accept only one user name that is missing a corresponding             |
| 30816 |                     | host name. The <i>qselect</i> utility shall accept only one user name per named host.                  |
| 30817 | <b>OPERANDS</b>     |                                                                                                        |
| 30818 |                     | None.                                                                                                  |
| 30819 | <b>STDIN</b>        |                                                                                                        |
| 30820 |                     | Not used.                                                                                              |
| 30821 | <b>INPUT FILES</b>  |                                                                                                        |
| 30822 |                     | None.                                                                                                  |

30823 **ENVIRONMENT VARIABLES**

30824 The following environment variables shall affect the execution of *qselect*:

30825 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30826 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 30827 Internationalization Variables for the precedence of internationalization variables  
 30828 used to determine the values of locale categories.)

30829 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30830 internationalization variables.

30831 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30832 characters (for example, single-byte as opposed to multi-byte characters in  
 30833 arguments).

30834 *LC\_MESSAGES*  
 30835 Determine the locale that should be used to affect the format and contents of  
 30836 diagnostic messages written to standard error.

30837 *LOGNAME* Determine the login name of the user.

30838 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is  
 30839 unset or null, an unspecified default timezone shall be used.

30840 **ASYNCHRONOUS EVENTS**

30841 Default.

30842 **STDOUT**

30843 The *qselect* utility shall write zero or more batch *job\_identifiers* to standard output.

30844 The *qselect* utility shall separate the batch *job\_identifiers* written to standard output by white  
 30845 space.

30846 The *qselect* utility shall write batch *job\_identifiers* in the following format:

30847 *sequence\_number.server\_name@server*

30848 **STDERR**

30849 The standard error shall be used only for diagnostic messages.

30850 **OUTPUT FILES**

30851 None.

30852 **EXTENDED DESCRIPTION**

30853 None.

30854 **EXIT STATUS**

30855 The following exit values shall be returned:

30856 0 Successful completion.

30857 >0 An error occurred.

30858 **CONSEQUENCES OF ERRORS**

30859 Default.

30860 **APPLICATION USAGE**

30861       None.

30862 **EXAMPLES**

30863       The following example shows how a user might use the *qselect* utility in conjunction with the  
 30864       *qdel* utility to delete all of his or her jobs in the queued state without affecting any jobs that are  
 30865       already running:

30866       `qdel $(qselect -s q)`

30867       or:

30868       `qselect -s q || xargs qdel`30869 **RATIONALE**

30870       The *qselect* utility allows users to acquire a list of job identifiers that match user-specified  
 30871       selection criteria. The list of identifiers returned by the *qselect* utility conforms to the syntax of  
 30872       the batch job identifier list processed by a utility such as *qmove*, *qdel*, and *qrls*. The *qselect* utility is  
 30873       thus a powerful tool for causing another batch system utility to act upon a set of jobs that match  
 30874       a list of selection criteria.

30875       The options of the *qselect* utility let the user apply a number of useful filters for selecting jobs.  
 30876       Each option further restricts the selection of jobs. Many of the selection options allow the  
 30877       specification of a relational operator. The FORTRAN-like syntax of the operator—that is,  
 30878       "*.lt.*"—was chosen rather than the C-like "*<=*" meta-characters.

30879       The **-a** option allows users to restrict the selected jobs to those that have been submitted (or  
 30880       altered) to wait until a particular time. The time period is determined by the argument of this  
 30881       option, which includes both a time and an operator—it is thus possible to select jobs waiting  
 30882       until a specific time, jobs waiting until after a certain time, or those waiting for a time before the  
 30883       specified time.

30884       The **-A** option allows users to restrict the selected jobs to those that have been submitted (or  
 30885       altered) to charge a particular account.

30886       The **-c** option allows users to restrict the selected jobs to those whose checkpointing interval  
 30887       falls within the specified range.

30888       The **-l** option allows users to select those jobs whose resource limits fall within the range  
 30889       indicated by the value of the option. For example, a user could select those jobs for which the  
 30890       CPU time limit is greater than two hours.

30891       The **-N** option allows users to select jobs by job name. For instance, all the parts of a task that  
 30892       have been divided in parallel jobs might be given the same name, and thus manipulated as a  
 30893       group by means of this option.

30894       The **-q** option allows users to select jobs in a specified queue.

30895       The **-r** option allows users to select only those jobs with a specified rerun criteria. For instance, a  
 30896       user might select only those jobs that can be rerun for use with the *qrerun* utility.

30897       The **-s** option allows users to select only those jobs that are in a certain state.

30898       The **-u** option allows users to select jobs that have been submitted to execute under a particular  
 30899       account.

30900       The selection criteria provided by the options of the *qselect* utility allow users to select jobs based  
 30901       on all the appropriate attributes that can be assigned to jobs by the *qsub* utility.

30902       Historically, the *qselect* utility has not been a part of existing practice; it is an improvement that  
 30903       has been introduced in this volume of IEEE Std 1003.1-2001.

30904 **FUTURE DIRECTIONS**

30905           None.

30906 **SEE ALSO**30907           *qdel*, *qrerun*, *qrls*, *qselect*, *qsub*, *touch*, Chapter 3 (on page 101)30908 **CHANGE HISTORY**

30909           Derived from IEEE Std 1003.2d-1994.

30910 **NAME**

30911           qsig — signal batch jobs

30912 **SYNOPSIS**30913 BE       qsig [-s *signal*] *job\_identifier* ...

30914

30915 **DESCRIPTION**

30916       To signal a batch job is to send a signal to the session leader of the batch job. A batch job is  
 30917       signaled by sending a request to the batch server that manages the batch job. The *qsig* utility is a  
 30918       user-accessible batch client that requests the signaling of a batch job.

30919       The *qsig* utility shall signal those batch jobs for which a batch *job\_identifier* is presented to the  
 30920       utility. The *qsig* utility shall not signal any batch jobs whose batch *job\_identifiers* are not  
 30921       presented to the utility.

30922       The *qsig* utility shall signal batch jobs in the order in which the corresponding batch  
 30923       *job\_identifiers* are presented to the utility. If the *qsig* utility fails to process a batch *job\_identifier*  
 30924       successfully, the utility shall proceed to process the remaining batch *job\_identifiers*, if any.

30925       The *qsig* utility shall signal batch jobs by sending a *Signal Job Request* to the batch server that  
 30926       manages the batch job.

30927       For each successfully processed batch *job\_identifier*, the *qsig* utility shall have received a  
 30928       completion reply to each *Signal Job Request* sent to a batch server at the time the utility exits.

30929 **OPTIONS**

30930       The *qsig* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 30931       12.2, Utility Syntax Guidelines.

30932       The following option shall be supported by the implementation:

30933       -s *signal*     Define the signal to be sent to the batch job.

30934       The *qsig* utility shall accept a *signal* option-argument that is either a symbolic  
 30935       signal name or an unsigned integer signal number (see the POSIX.1-1990 standard,  
 30936       Section 3.3.1.1). The *qsig* utility shall accept signal names for which the SIG prefix  
 30937       has been omitted.

30938       If the *signal* option-argument is a signal name, the *qsig* utility shall send that name.

30939       If the *signal* option-argument is a number, the *qsig* utility shall send the signal  
 30940       value represented by the number.

30941       If the -s option is not presented to the *qsig* utility, the utility shall send the signal  
 30942       SIGTERM to each signaled batch job.

30943 **OPERANDS**

30944       The *qsig* utility shall accept one or more operands that conform to the syntax for a batch  
 30945       *job\_identifier* (see Section 3.3.1 (on page 122)).

30946 **STDIN**

30947       Not used.

30948 **INPUT FILES**

30949       None.

## 30950 ENVIRONMENT VARIABLES

30951 The following environment variables shall affect the execution of *qsig*:

30952 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30953 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 30954 Internationalization Variables for the precedence of internationalization variables  
 30955 used to determine the values of locale categories.)

30956 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30957 internationalization variables.

30958 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30959 characters (for example, single-byte as opposed to multi-byte characters in  
 30960 arguments).

30961 *LC\_MESSAGES*  
 30962 Determine the locale that should be used to affect the format and contents of  
 30963 diagnostic messages written to standard error.

30964 *LOGNAME* Determine the login name of the user.

## 30965 ASYNCHRONOUS EVENTS

30966 Default.

## 30967 STDOUT

30968 An implementation of the *qsig* utility may write informative messages to standard output.

## 30969 STDERR

30970 The standard error shall be used only for diagnostic messages.

## 30971 OUTPUT FILES

30972 None.

## 30973 EXTENDED DESCRIPTION

30974 None.

## 30975 EXIT STATUS

30976 The following exit values shall be returned:

30977 0 Successful completion.

30978 >0 An error occurred.

## 30979 CONSEQUENCES OF ERRORS

30980 In addition to the default behavior, the *qsig* utility shall not be required to write a diagnostic  
 30981 message to standard error when the error reply received from a batch server indicates that the  
 30982 batch *job\_identifier* does not exist on the server. Whether or not the *qsig* utility waits to output the  
 30983 diagnostic message while attempting to locate the batch job on other servers is implementation-  
 30984 defined.

## 30985 APPLICATION USAGE

30986 None.

## 30987 EXAMPLES

30988 None.

## 30989 RATIONALE

30990 The *qsig* utility allows users to signal batch jobs.

30991 A user may be unable to signal a batch job with the *kill* utility of the operating system for a  
 30992 number of reasons. First, the process ID of the batch job may be unknown to the user. Second,



- 30993 the processes of the batch job may be on a remote node. However, by virtue of communication  
30994 between batch nodes, the *qsig* utility can arrange for the signaling of a process.
- 30995 Because a batch job that is not running cannot be signaled, and because the signal may not  
30996 terminate the batch job, the *qsig* utility is not a substitute for the *qdel* utility.
- 30997 The options of the *qsig* utility allow the user to specify the signal that is to be sent to the batch  
30998 job.
- 30999 The **-s** option allows users to specify a signal by name or by number, and thus override the  
31000 default signal. The POSIX.1-1990 standard defines signals by both name and number.
- 31001 The *qsig* utility is a new utility, *vis-a-vis* existing practice; it has been defined in this volume of  
31002 IEEE Std 1003.1-2001 in response to user-perceived shortcomings in existing practice.
- 31003 **FUTURE DIRECTIONS**
- 31004 None.
- 31005 **SEE ALSO**
- 31006 Chapter 3 (on page 101), *kill*, *qdel*
- 31007 **CHANGE HISTORY**
- 31008 Derived from IEEE Std 1003.2d-1994.
- 31009 **Issue 6**
- 31010 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

31011 **NAME**

31012 qstat — show status of batch jobs

31013 **SYNOPSIS**31014 BE qstat [-f] *job\_identifier* ...31015 qstat -Q [-f] *destination* ...31016 qstat -B [-f] *server\_name* ...

31017

31018 **DESCRIPTION**

31019 The status of a batch job, batch queue, or batch server is obtained by a request to the server. The  
 31020 *qstat* utility is a user-accessible batch client that requests the status of one or more batch jobs,  
 31021 batch queues, or servers, and writes the status information to standard output.

31022 For each successfully processed batch *job\_identifier*, the *qstat* utility shall display information  
 31023 about the corresponding batch job.

31024 For each successfully processed destination, the *qstat* utility shall display information about the  
 31025 corresponding batch queue.

31026 For each successfully processed server name, the *qstat* utility shall display information about the  
 31027 corresponding server.

31028 The *qstat* utility shall acquire batch job status information by sending a *Job Status Request* to a  
 31029 batch server. The *qstat* utility shall acquire batch queue status information by sending a *Queue*  
 31030 *Status Request* to a batch server. The *qstat* utility shall acquire server status information by  
 31031 sending a *Server Status Request* to a batch server.

31032 **OPTIONS**

31033 The *qstat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 31034 12.2, Utility Syntax Guidelines.

31035 The following options shall be supported by the implementation:

31036 **-f** Specify that a full display is produced.

31037 The minimum contents of a full display are specified in the STDOUT section.

31038 Additional contents and format of a full display are implementation-defined.

31039 **-Q** Specify that the operand is a destination.

31040 The *qstat* utility shall display information about each batch queue at each  
 31041 destination identified as an operand.

31042 **-B** Specify that the operand is a server name.

31043 The *qstat* utility shall display information about each server identified as an  
 31044 operand.

31045 **OPERANDS**

31046 If the **-Q** option is presented to the *qstat* utility, the utility shall accept one or more operands that  
 31047 conform to the syntax for a destination (see Section 3.3.2 (on page 123)).

31048 If the **-B** option is presented to the *qstat* utility, the utility shall accept one or more *server\_name*  
 31049 operands.

31050 If neither the **-B** nor the **-Q** option is presented to the *qstat* utility, the utility shall accept one or  
 31051 more operands that conform to the syntax for a batch *job\_identifier* (see Section 3.3.1 (on page  
 31052 122)).

31053 **STDIN**

31054 Not used.

31055 **INPUT FILES**

31056 None.

31057 **ENVIRONMENT VARIABLES**31058 The following environment variables shall affect the execution of *qstat*:31059 *HOME* Determine the pathname of the user's home directory.

31060 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 31061 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 31062 Internationalization Variables for the precedence of internationalization variables  
 31063 used to determine the values of locale categories.)

31064 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 31065 internationalization variables.

31066 *LC\_COLLATE*

31067 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 31068 character collating elements within regular expressions.

31069 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 31070 characters (for example, single-byte as opposed to multi-byte characters in  
 31071 arguments).

31072 *LC\_MESSAGES*

31073 Determine the locale that should be used to affect the format and contents of  
 31074 diagnostic messages written to standard error.

31075 *LC\_NUMERIC*

31076 Determine the locale for selecting the radix character used when writing floating-  
 31077 point formatted output.

31078 **ASYNCHRONOUS EVENTS**

31079 Default.

31080 **STDOUT**

31081 If an operand presented to the *qstat* utility is a batch *job\_identifier* and the **-f** option is not  
 31082 specified, the *qstat* utility shall display the following items on a single line, in the stated order,  
 31083 with white space between each item, for each successfully processed operand:

- 31084 • The batch *job\_identifier*
- 31085 • The batch job name
- 31086 • The *Job\_Owner* attribute
- 31087 • The CPU time used by the batch job
- 31088 • The batch job state
- 31089 • The batch job location

31090 If an operand presented to the *qstat* utility is a batch *job\_identifier* and the **-f** option is specified,  
 31091 the *qstat* utility shall display the following items for each success fully processed operand:

- 31092 • The batch *job\_identifier*
- 31093 • The batch job name

- 31094 • The *Job\_Owner* attribute
  - 31095 • The execution user ID
  - 31096 • The CPU time used by the batch job
  - 31097 • The batch job state
  - 31098 • The batch job location
  - 31099 • Additional implementation-defined information, if any, about the batch job or batch queue
- 31100 If an operand presented to the *qstat* utility is a destination, the **-Q** option is specified, and the **-f**  
 31101 option is not specified, the *qstat* utility shall display the following items on a single line, in the  
 31102 stated order, with white space between each item, for each successfully processed operand:
- 31103 • The batch queue name
  - 31104 • The maximum number of batch jobs that shall be run in the batch queue concurrently
  - 31105 • The total number of batch jobs in the batch queue
  - 31106 • The status of the batch queue
  - 31107 • For each state, the number of batch jobs in that state in the batch queue and the name of the  
 31108 state
  - 31109 • The type of batch queue (execution or routing)
- 31110 If the operands presented to the *qstat* utility are destinations, the **-Q** option is specified, and the  
 31111 **-f** option is specified, the *qstat* utility shall display the following items for each successfully  
 31112 processed operand:
- 31113 • The batch queue name
  - 31114 • The maximum number of batch jobs that shall be run in the batch queue concurrently
  - 31115 • The total number of batch jobs in the batch queue
  - 31116 • The status of the batch queue
  - 31117 • For each state, the number of batch jobs in that state in the batch queue and the name of the  
 31118 state
  - 31119 • The type of batch queue (execution or routing)
  - 31120 • Additional implementation-defined information, if any, about the batch queue
- 31121 If the operands presented to the *qstat* utility are batch server names, the **-B** option is specified,  
 31122 and the **-f** option is not specified, the *qstat* utility shall display the following items on a single  
 31123 line, in the stated order, with white space between each item, for each successfully processed  
 31124 operand:
- 31125 • The batch server name
  - 31126 • The maximum number of batch jobs that shall be run in the batch queue concurrently
  - 31127 • The total number of batch jobs managed by the batch server
  - 31128 • The status of the batch server
  - 31129 • For each state, the number of batch jobs in that state and the name of the state
- 31130 If the operands presented to the *qstat* utility are server names, the **-B** option is specified, and the  
 31131 **-f** option is specified, the *qstat* utility shall display the following items for each successfully  
 31132 processed operand:

- 31133
  - The server name
- 31134
  - The maximum number of batch jobs that shall be run in the batch queue concurrently
- 31135
  - The total number of batch jobs managed by the server
- 31136
  - The status of the server
- 31137
  - For each state, the number of batch jobs in that state and the name of the state
- 31138
  - Additional implementation-defined information, if any, about the server
- 31139 **STDERR**
- 31140 The standard error shall be used only for diagnostic messages.
- 31141 **OUTPUT FILES**
- 31142 None.
- 31143 **EXTENDED DESCRIPTION**
- 31144 None.
- 31145 **EXIT STATUS**
- 31146 The following exit values shall be returned:
- 31147 0 Successful completion.
- 31148 >0 An error occurred.
- 31149 **CONSEQUENCES OF ERRORS**
- 31150 In addition to the default behavior, the *qstat* utility shall not be required to write a diagnostic
- 31151 message to standard error when the error reply received from a batch server indicates that the
- 31152 batch *job\_identifier* does not exist on the server. Whether or not the *qstat* utility waits to output
- 31153 the diagnostic message while attempting to locate the batch job on other servers is
- 31154 implementation-defined.
- 31155 **APPLICATION USAGE**
- 31156 None.
- 31157 **EXAMPLES**
- 31158 None.
- 31159 **RATIONALE**
- 31160 The *qstat* utility allows users to display the status of jobs and list the batch jobs in queues.
- 31161 The operands of the *qstat* utility may be either job identifiers, queues (specified as destination
- 31162 identifiers), or batch server names. The **-Q** and **-B** options, or absence thereof, indicate the
- 31163 nature of the operands.
- 31164 The other options of the *qstat* utility allow the user to control the amount of information
- 31165 displayed and the format in which it is displayed. Should a user wish to display the status of a
- 31166 set of jobs that match a selection criteria, the *qselect* utility may be used to acquire such a list.
- 31167 The **-f** option allows users to request a “full” display in an implementation-defined format.
- 31168 Historically, the *qstat* utility has been a part of the NQS and its derivatives, the existing practice
- 31169 on which it is based.
- 31170 **FUTURE DIRECTIONS**
- 31171 None.

31172 **SEE ALSO**31173 Chapter 3 (on page 101), *qselect*31174 **CHANGE HISTORY**

31175 Derived from IEEE Std 1003.2d-1994.

31176 **Issue 6**31177 IEEE PASC Interpretation 1003.2 #191 is applied, removing the following ENVIRONMENT  
31178 VARIABLES listed as affecting *qstat*: *COLUMNS*, *LINES*, *LOGNAME*, *TERM*, and *TZ*.31179 The *LC\_TIME* entry is also removed from the ENVIRONMENT VARIABLES section.

31180 **NAME**

31181           qsub — submit a script

31182 **SYNOPSIS**

```

31183 BE qsub [-a date_time] [-A account_string] [-c interval]
31184 [-C directive_prefix] [-e path_name] [-h] [-j join_list] [-k keep_list]
31185 [-m mail_options] [-M mail_list] [-N name]
31186 [-o path_name] [-p priority] [-q destination] [-r y|n]
31187 [-S path_name_list] [-u user_list] [-v variable_list] [-V]
31188 [-z] [script]
31189

```

31190 **DESCRIPTION**

31191       To submit a script is to create a batch job that executes the script. A script is submitted by a  
 31192       request to a batch server. The *qsub* utility is a user-accessible batch client that submits a script.

31193       Upon successful completion, the *qsub* utility shall have created a batch job that will execute the  
 31194       submitted script.

31195       The *qsub* utility shall submit a script by sending a *Queue Job Request* to a batch server.

31196       The *qsub* utility shall place the value of the following environment variables in the *Variable\_List*  
 31197       attribute of the batch job: *HOME*, *LANG*, *LOGNAME*, *PATH*, *MAIL*, *SHELL*, and *TZ*. The name  
 31198       of the environment variable shall be the current name prefixed with the string *PBS\_O\_*.

31199       **Note:**     If the current value of the *HOME* variable in the environment space of the *qsub* utility is  
 31200       */aa/bb/cc*, then *qsub* shall place *PBS\_O\_HOME=/aa/bb/cc* in the *Variable\_List* attribute of the  
 31201       batch job.

31202       In addition to the variables described above, the *qsub* utility shall add the following variables  
 31203       with the indicated values to the variable list:

31204       *PBS\_O\_WORKDIR*     The absolute path of the current working directory of the *qsub* utility  
 31205       process.

31206       *PBS\_O\_HOST*        The name of the host on which the *qsub* utility is running.

31207 **OPTIONS**

31208       The *qsub* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 31209       12.2, Utility Syntax Guidelines.

31210       The following options shall be supported by the implementation:

31211       **-a *date\_time***    Define the time at which a batch job becomes eligible for execution.

31212       The *qsub* utility shall accept an option-argument that conforms to the syntax of the  
 31213       *time* operand of the *touch* utility.

**Table 4-18** Environment Variable Values (Utilities)

| Variable Name        | Value at qsub Time        |
|----------------------|---------------------------|
| <i>PBS_O_HOME</i>    | <i>HOME</i>               |
| <i>PBS_O_HOST</i>    | Client host name          |
| <i>PBS_O_LANG</i>    | <i>LANG</i>               |
| <i>PBS_O_LOGNAME</i> | <i>LOGNAME</i>            |
| <i>PBS_O_PATH</i>    | <i>PATH</i>               |
| <i>PBS_O_MAIL</i>    | <i>MAIL</i>               |
| <i>PBS_O_SHELL</i>   | <i>SHELL</i>              |
| <i>PBS_O_TZ</i>      | <i>TZ</i>                 |
| <i>PBS_O_WORKDIR</i> | Current working directory |

**Note:** The server that initiates execution of the batch job will add other variables to the batch job's environment; see Section 3.2.2.1 (on page 106).

The *qsub* utility shall set the *Execution\_Time* attribute of the batch job to the number of seconds since the Epoch that is equivalent to the local time expressed by the value of the *date\_time* option-argument. The Epoch is defined in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.149, Epoch.

If the **-a** option is not presented to the *qsub* utility, the utility shall set the *Execution\_Time* attribute of the batch job to a time (number of seconds since the Epoch) that is earlier than the time at which the utility exits.

**-A** *account\_string*

Define the account to which the resource consumption of the batch job should be charged.

The syntax of the *account\_string* option-argument is unspecified.

The *qsub* utility shall set the *Account\_Name* attribute of the batch job to the value of the *account\_string* option-argument.

If the **-A** option is not presented to the *qsub* utility, the utility shall omit the *Account\_Name* attribute from the attributes of the batch job.

**-c** *interval*

Define whether the batch job should be checkpointed, and if so, how often.

The *qsub* utility shall accept a value for the interval option-argument that is one of the following:

**n** No checkpointing shall be performed on the batch job (NO\_CHECKPOINT).

**s** Checkpointing shall be performed only when the batch server is shut down (CHECKPOINT\_AT\_SHUTDOWN).

**c** Automatic periodic checkpointing shall be performed at the *Minimum\_Cpu\_Interval* attribute of the batch queue, in units of CPU minutes (CHECKPOINT\_AT\_MIN\_CPU\_INTERVAL).

**c=minutes** Automatic periodic checkpointing shall be performed every *minutes* of CPU time, or every *Minimum\_Cpu\_Interval* minutes, whichever is greater. The *minutes* argument shall conform to the syntax for unsigned integers and shall be greater than zero.

The *qsub* utility shall set the *Checkpoint* attribute of the batch job to the value of the *interval* option-argument.



31258 If the **-c** option is not presented to the *qsub* utility, the utility shall set the  
 31259 *Checkpoint* attribute of the batch job to the single character 'u'  
 31260 (*CHECKPOINT\_UNSPECIFIED*).

31261 **-C directive\_prefix**  
 31262 Define the prefix that declares a directive to the *qsub* utility within the script.

31263 The *directive\_prefix* is not a batch job attribute; it affects the behavior of the *qsub*  
 31264 utility.

31265 If the **-C** option is presented to the *qsub* utility, and the value of the *directive\_prefix*  
 31266 option-argument is the null string, the utility shall not scan the script file for  
 31267 directives. If the **-C** option is not presented to the *qsub* utility, then the value of the  
 31268 *PBS\_DPREFIX* environment variable is used. If the environment variable is not  
 31269 defined, then #PBS encoded in the portable character set is the default.

31270 **-e path\_name**  
 31271 Define the path to be used for the standard error stream of the batch job.

31272 The *qsub* utility shall accept a *path\_name* option-argument which can be preceded  
 31273 by a host name element of the form *hostname*:.  
 31274 If the *path\_name* option-argument constitutes an absolute pathname, the *qsub*  
 31275 utility shall set the *Error\_Path* attribute of the batch job to the value of the  
 31276 *path\_name* option-argument.

31277 If the *path\_name* option-argument constitutes a relative pathname and no host  
 31278 name element is specified, the *qsub* utility shall set the *Error\_Path* attribute of the  
 31279 batch job to the value of the absolute pathname derived by expanding the  
 31280 *path\_name* option-argument relative to the current directory of the process  
 31281 executing *qsub*.

31282 If the *path\_name* option-argument constitutes a relative pathname and a host name  
 31283 element is specified, the *qsub* utility shall set the *Error\_Path* attribute of the batch  
 31284 job to the value of the *path\_name* option-argument without expansion. The host  
 31285 name element shall be included.

31286 If the *path\_name* option-argument does not include a host name element, the *qsub*  
 31287 utility shall prefix the pathname with *hostname*:, where *hostname* is the name of the  
 31288 host upon which the *qsub* utility is being executed.

31289 If the **-e** option is not presented to the *qsub* utility, the utility shall set the  
 31290 *Error\_Path* attribute of the batch job to the host name and path of the current  
 31291 directory of the submitting process and the default filename.

31292 The default filename for standard error has the following format:  
 31293 *job\_name.esquence\_number*

31294 **-h** Specify that a USER hold is applied to the batch job.

31295 The *qsub* utility shall set the value of the *Hold\_Types* attribute of the batch job to the  
 31296 value USER.

31297 If the **-h** option is not presented to the *qsub* utility, the utility shall set the  
 31298 *Hold\_Types* attribute of the batch job to the value NO\_HOLD.

31299 **-j join\_list** Define which streams of the batch job are to be merged. The *qsub* **-j** option shall  
 31300 accept a value for the *join\_list* option-argument that is a string of alphanumeric  
 31301 characters in the portable character set (see the Base Definitions volume of

31302 IEEE Std 1003.1-2001, Section 6.1, Portable Character Set).

31303 The *qsub* utility shall accept a *join\_list* option-argument that consists of one or  
 31304 more of the characters 'e' and 'o', or the single character 'n'.

31305 All of the other batch job output streams specified will be merged into the output  
 31306 stream represented by the character listed first in the *join\_list* option-argument.

31307 For each unique character in the *join\_list* option-argument, the *qsub* utility shall  
 31308 add a value to the *Join\_Path* attribute of the batch job as follows, each representing  
 31309 a different batch job stream to join:

- 31310 e The standard error of the batch job (JOIN\_STD\_ERROR).
- 31311 o The standard output of the batch job (JOIN\_STD\_OUTPUT).

31312 An existing *Join\_Path* attribute can be cleared by the following join type:

31313 n NO\_JOIN

31314 If 'n' is specified, then no files are joined. The *qsub* utility shall consider it an error  
 31315 if any join type other than 'n' is combined with join type 'n'.

31316 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or  
 31317 'n' within the *join\_list* option-argument. The *qsub* utility shall permit the  
 31318 repetition of characters, but shall not assign additional meaning to the repeated  
 31319 characters.

31320 An implementation may define other join types. The conformance document for an  
 31321 implementation shall describe any additional batch job streams, how they are  
 31322 specified, their internal behavior, and how they affect the behavior of the utility.

31323 If the **-j** option is not presented to the *qsub* utility, the utility shall set the value of  
 31324 the *Join\_Path* attribute of the batch job to NO\_JOIN.

31325 **-k keep\_list** Define which output of the batch job to retain on the execution host.

31326 The *qsub* **-k** option shall accept a value for the *keep\_list* option-argument that is a  
 31327 string of alphanumeric characters in the portable character set (see the Base  
 31328 Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set).

31329 The *qsub* utility shall accept a *keep\_list* option-argument that consists of one or  
 31330 more of the characters 'e' and 'o', or the single character 'n'.

31331 For each unique character in the *keep\_list* option-argument, the *qsub* utility shall  
 31332 add a value to the *Keep\_Files* attribute of the batch job as follows, each representing  
 31333 a different batch job stream to keep:

- 31334 e The standard error of the batch job (KEEP\_STD\_ERROR).
- 31335 o The standard output of the batch job (KEEP\_STD\_OUTPUT).

31336 If both 'e' and 'o' are specified, then both files are retained. An existing  
 31337 *Keep\_Files* attribute can be cleared by the following keep type:

31338 n NO\_KEEP

31339 If 'n' is specified, then no files are retained. The *qsub* utility shall consider it an  
 31340 error if any keep type other than 'n' is combined with keep type 'n'.

31341 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or  
 31342 'n' within the *keep\_list* option-argument. The *qsub* utility shall permit the  
 31343 repetition of characters, but shall not assign additional meaning to the repeated

31344 characters.

31345 An implementation may define other keep types. The conformance document for  
 31346 an implementation shall describe any additional keep types, how they are  
 31347 specified, their internal behavior, and how they affect the behavior of the utility. If  
 31348 the **-k** option is not presented to the *qsub* utility, the utility shall set the *Keep\_Files*  
 31349 attribute of the batch job to the value NO\_KEEP.

31350 **-m mail\_options**

31351 Define the points in the execution of the batch job at which the batch server that  
 31352 manages the batch job shall send mail about a change in the state of the batch job.

31353 The *qsub -m* option shall accept a value for the *mail\_options* option-argument that  
 31354 is a string of alphanumeric characters in the portable character set (see the Base  
 31355 Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set).

31356 The *qsub* utility shall accept a value for the *mail\_options* option-argument that is a  
 31357 string of one or more of the characters 'e', 'b', and 'a', or the single character  
 31358 'n'.

31359 For each unique character in the *mail\_options* option-argument, the *qsub* utility shall  
 31360 add a value to the *Mail\_Users* attribute of the batch job as follows, each  
 31361 representing a different time during the life of a batch job at which to send mail:

31362 e MAIL\_AT\_EXIT

31363 b MAIL\_AT\_BEGINNING

31364 a MAIL\_AT\_ABORT

31365 If any of these characters are duplicated in the *mail\_options* option-argument, the  
 31366 duplicates shall be ignored.

31367 An existing *Mail\_Points* attribute can be cleared by the following mail type:

31368 n NO\_MAIL

31369 If 'n' is specified, then mail is not sent. The *qsub* utility shall consider it an error if  
 31370 any mail type other than 'n' is combined with mail type 'n'.

31371 Strictly conforming applications shall not repeat any of the characters 'e', 'b',  
 31372 'a', or 'n' within the *mail\_options* option-argument.

31373 The *qsub* utility shall permit the repetition of characters, but shall not assign  
 31374 additional meaning to the repeated characters. An implementation may define  
 31375 other mail types. The conformance document for an implementation shall describe  
 31376 any additional mail types, how they are specified, their internal behavior, and how  
 31377 they affect the behavior of the utility.

31378 If the **-m** option is not presented to the *qsub* utility, the utility shall set the  
 31379 *Mail\_Points* attribute to the value MAIL\_AT\_ABORT.

31380 **-M mail\_list** Define the list of users to which a batch server that executes the batch job shall  
 31381 send mail, if the server sends mail about the batch job.

31382 The syntax of the *mail\_list* option-argument is unspecified.

31383 If the implementation of the *qsub* utility uses a name service to locate users, the  
 31384 utility should accept the syntax used by the name service.

31385 If the implementation of the *qsub* utility does not use a name service to locate  
 31386 users, the implementation should accept the following syntax for user names:

|       |                     |                                                                                                        |
|-------|---------------------|--------------------------------------------------------------------------------------------------------|
| 31387 |                     | <i>mail_address</i> [,, <i>mail_address</i> ,, ...]                                                    |
| 31388 |                     | The interpretation of <i>mail_address</i> is implementation-defined.                                   |
| 31389 |                     | The <i>qsub</i> utility shall set the <i>Mail_Users</i> attribute of the batch job to the value of the |
| 31390 |                     | <i>mail_list</i> option-argument.                                                                      |
| 31391 |                     | If the <b>-M</b> option is not presented to the <i>qsub</i> utility, the utility shall place only the  |
| 31392 |                     | user name and host name for the current process in the <i>Mail_Users</i> attribute of the              |
| 31393 |                     | batch job.                                                                                             |
| 31394 | <b>-N name</b>      | Define the name of the batch job.                                                                      |
| 31395 |                     | The <i>qsub</i> <b>-N</b> option shall accept a value for the <i>name</i> option-argument that is a    |
| 31396 |                     | string of up to 15 alphanumeric characters in the portable character set (see the                      |
| 31397 |                     | Base Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character                       |
| 31398 |                     | Set) where the first character is alphabetic.                                                          |
| 31399 |                     | The <i>qsub</i> utility shall set the value of the <i>Job_Name</i> attribute of the batch job to the   |
| 31400 |                     | value of the <i>name</i> option-argument.                                                              |
| 31401 |                     | If the <b>-N</b> option is not presented to the <i>qsub</i> utility, the utility shall set the         |
| 31402 |                     | <i>Job_Name</i> attribute of the batch job to the name of the <i>script</i> argument from which        |
| 31403 |                     | the directory specification if any, has been removed.                                                  |
| 31404 |                     | If the <b>-N</b> option is not presented to the <i>qsub</i> utility, and the script is read from       |
| 31405 |                     | standard input, the utility shall set the <i>Job_Name</i> attribute of the batch job to the            |
| 31406 |                     | value STDIN.                                                                                           |
| 31407 | <b>-o path_name</b> |                                                                                                        |
| 31408 |                     | Define the path for the standard output of the batch job.                                              |
| 31409 |                     | The <i>qsub</i> utility shall accept a <i>path_name</i> option-argument that conforms to the           |
| 31410 |                     | syntax of the <i>path_name</i> element defined in the System Interfaces volume of                      |
| 31411 |                     | IEEE Std 1003.1-2001, which can be preceded by a host name element of the form                         |
| 31412 |                     | <i>hostname</i> ..                                                                                     |
| 31413 |                     | If the <i>path_name</i> option-argument constitutes an absolute pathname, the <i>qsub</i>              |
| 31414 |                     | utility shall set the <i>Output_Path</i> attribute of the batch job to the value of the                |
| 31415 |                     | <i>path_name</i> option-argument without expansion.                                                    |
| 31416 |                     | If the <i>path_name</i> option-argument constitutes a relative pathname and no host                    |
| 31417 |                     | name element is specified, the <i>qsub</i> utility shall set the <i>Output_Path</i> attribute of the   |
| 31418 |                     | batch job to the pathname derived by expanding the value of the <i>path_name</i>                       |
| 31419 |                     | option-argument relative to the current directory of the process executing the <i>qsub</i> .           |
| 31420 |                     | If the <i>path_name</i> option-argument constitutes a relative pathname and a host name                |
| 31421 |                     | element is specified, the <i>qsub</i> utility shall set the <i>Output_Path</i> attribute of the batch  |
| 31422 |                     | job to the value of the <i>path_name</i> option-argument without expansion.                            |
| 31423 |                     | If the <i>path_name</i> option-argument does not specify a host name element, the <i>qsub</i>          |
| 31424 |                     | utility shall prefix the pathname with <i>hostname</i> ., where <i>hostname</i> is the name of the     |
| 31425 |                     | host upon which the <i>qsub</i> utility is executing.                                                  |
| 31426 |                     | If the <b>-o</b> option is not presented to the <i>qsub</i> utility, the utility shall set the         |
| 31427 |                     | <i>Output_Path</i> attribute of the batch job to the host name and path of the current                 |
| 31428 |                     | directory of the submitting process and the default filename.                                          |
| 31429 |                     | The default filename for standard output has the following format:                                     |

|       |                                 |                                                                                                                                                                                                                                      |
|-------|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 31430 |                                 | <i>job_name.osequence_number</i>                                                                                                                                                                                                     |
| 31431 | <b>-p</b> <i>priority</i>       | Define the priority the batch job should have relative to other batch jobs owned by the batch server.                                                                                                                                |
| 31432 |                                 |                                                                                                                                                                                                                                      |
| 31433 |                                 | The <i>qsub</i> utility shall set the <i>Priority</i> attribute of the batch job to the value of the <i>priority</i> option-argument.                                                                                                |
| 31434 |                                 |                                                                                                                                                                                                                                      |
| 31435 |                                 | If the <b>-p</b> option is not presented to the <i>qsub</i> utility, the value of the <i>Priority</i> attribute is implementation-defined.                                                                                           |
| 31436 |                                 |                                                                                                                                                                                                                                      |
| 31437 |                                 | The <i>qsub</i> utility shall accept a value for the <i>priority</i> option-argument that conforms to the syntax for signed decimal integers, and which is not less than -1 024 and not greater than 1 023.                          |
| 31438 |                                 |                                                                                                                                                                                                                                      |
| 31439 |                                 |                                                                                                                                                                                                                                      |
| 31440 | <b>-q</b> <i>destination</i>    |                                                                                                                                                                                                                                      |
| 31441 |                                 | Define the destination of the batch job.                                                                                                                                                                                             |
| 31442 |                                 | The destination is not a batch job attribute; it determines the batch server, and possibly the batch queue, to which the <i>qsub</i> utility batch queues the batch job.                                                             |
| 31443 |                                 |                                                                                                                                                                                                                                      |
| 31444 |                                 | The <i>qsub</i> utility shall submit the script to the batch server named by the <i>destination</i> option-argument or the server that owns the batch queue named in the <i>destination</i> option-argument.                         |
| 31445 |                                 |                                                                                                                                                                                                                                      |
| 31446 |                                 |                                                                                                                                                                                                                                      |
| 31447 |                                 | The <i>qsub</i> utility shall accept an option-argument for the <b>-q</b> option that conforms to the syntax for a destination (see Section 3.3.2 (on page 123)).                                                                    |
| 31448 |                                 |                                                                                                                                                                                                                                      |
| 31449 |                                 | If the <b>-q</b> option is not presented to the <i>qsub</i> utility, the <i>qsub</i> utility shall submit the batch job to the default destination. The mechanism for determining the default destination is implementation-defined. |
| 31450 |                                 |                                                                                                                                                                                                                                      |
| 31451 |                                 |                                                                                                                                                                                                                                      |
| 31452 | <b>-r</b> <i>y   n</i>          | Define whether the batch job is rerunnable.                                                                                                                                                                                          |
| 31453 |                                 | If the value of the option-argument is <i>y</i> , the <i>qsub</i> utility shall set the <i>Rerunable</i> attribute of the batch job to TRUE.                                                                                         |
| 31454 |                                 |                                                                                                                                                                                                                                      |
| 31455 |                                 | If the value of the option-argument is <i>n</i> , the <i>qsub</i> utility shall set the <i>Rerunable</i> attribute of the batch job to FALSE.                                                                                        |
| 31456 |                                 |                                                                                                                                                                                                                                      |
| 31457 |                                 | If the <b>-r</b> option is not presented to the <i>qsub</i> utility, the utility shall set the <i>Rerunable</i> attribute of the batch job to TRUE.                                                                                  |
| 31458 |                                 |                                                                                                                                                                                                                                      |
| 31459 | <b>-S</b> <i>path_name_list</i> |                                                                                                                                                                                                                                      |
| 31460 |                                 | Define the pathname to the shell under which the batch job is to execute.                                                                                                                                                            |
| 31461 |                                 | The <i>qsub</i> utility shall accept a <i>path_name_list</i> option-argument that conforms to the following syntax:                                                                                                                  |
| 31462 |                                 |                                                                                                                                                                                                                                      |
| 31463 |                                 | <i>pathname</i> [@ <i>host</i> ] [, , <i>pathname</i> [@ <i>host</i> ] , , . . .]                                                                                                                                                    |
| 31464 |                                 | The <i>qsub</i> utility shall allow only one pathname for a given host name. The <i>qsub</i> utility shall allow only one pathname that is missing a corresponding host name.                                                        |
| 31465 |                                 |                                                                                                                                                                                                                                      |
| 31466 |                                 | The <i>qsub</i> utility shall add a value to the <i>Shell_Path_List</i> attribute of the batch job for each entry in the <i>path_name_list</i> option-argument.                                                                      |
| 31467 |                                 |                                                                                                                                                                                                                                      |
| 31468 |                                 | If the <b>-S</b> option is not presented to the <i>qsub</i> utility, the utility shall set the <i>Shell_Path_List</i> attribute of the batch job to the null string.                                                                 |
| 31469 |                                 |                                                                                                                                                                                                                                      |
| 31470 |                                 | The conformance document for an implementation shall describe the mechanism used to set the default shell and determine the current value of the default shell.                                                                      |
| 31471 |                                 |                                                                                                                                                                                                                                      |

31472 An implementation shall provide a means for the installation to set the default  
 31473 shell to the login shell of the user under which the batch job is to execute. See  
 31474 Section 3.3.3 (on page 123) for a means of removing *keyword=value* (and  
 31475 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

31476     **-u *user\_list*** Define the user name under which the batch job is to execute.

31477 The *qsub* utility shall accept a *user\_list* option-argument that conforms to the  
 31478 following syntax:

31479     *username*[@*host*] [, , *username*[@*host*] , , ...]

31480 The *qsub* utility shall accept only one user name that is missing a corresponding  
 31481 host name. The *qsub* utility shall accept only one user name per named host.

31482 The *qsub* utility shall add a value to the *User\_List* attribute of the batch job for each  
 31483 entry in the *user\_list* option-argument.

31484 If the **-u** option is not presented to the *qsub* utility, the utility shall set the *User\_List*  
 31485 attribute of the batch job to the user name from which the utility is executing. See  
 31486 Section 3.3.3 (on page 123) for a means of removing *keyword=value* (and  
 31487 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

31488     **-v *variable\_list***

31489 Add to the list of variables that are exported to the session leader of the batch job.

31490 A *variable\_list* is a set of strings of either the form <*variable*> or <*variable=value*>,  
 31491 delimited by commas.

31492 If the **-v** option is presented to the *qsub* utility, the utility shall also add, to the  
 31493 environment *Variable\_List* attribute of the batch job, every variable named in the  
 31494 environment *variable\_list* option-argument and, optionally, values of specified  
 31495 variables.

31496 If a value is not provided on the command line, the *qsub* utility shall set the value  
 31497 of each variable in the environment *Variable\_List* attribute of the batch job to the  
 31498 value of the corresponding environment variable for the process in which the  
 31499 utility is executing; see Table 4-18 (on page 804).

31500 A conforming application shall not repeat a variable in the environment  
 31501 *variable\_list* option-argument.

31502 The *qsub* utility shall not repeat a variable in the environment *Variable\_List*  
 31503 attribute of the batch job. See Section 3.3.3 (on page 123) for a means of removing  
 31504 *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented  
 31505 batch job attributes.

31506     **-V**

31507 Specify that all of the environment variables of the process are exported to the  
 context of the batch job.

31508 The *qsub* utility shall place every environment variable in the process in which the  
 31509 utility is executing in the list and shall set the value of each variable in the attribute  
 31510 to the value of that variable in the process.

31511     **-z**

31512 Specify that the utility does not write the batch *job\_identifier* of the created batch  
 job to standard output.

31513 If the **-z** option is presented to the *qsub* utility, the utility shall not write the batch  
 31514 *job\_identifier* of the created batch job to standard output.

31515 If the **-z** option is not presented to the *qsub* utility, the utility shall write the  
 31516 identifier of the created batch job to standard output.

### 31517 OPERANDS

31518 The *qsub* utility shall accept a *script* operand that indicates the path to the script of the batch job.

31519 If the *script* operand is not presented to the *qsub* utility, or if the operand is the single-character  
 31520 string ' - ', the utility shall read the script from standard input.

31521 If the script represents a partial path, the *qsub* utility shall expand the path relative to the current  
 31522 directory of the process executing the utility.

### 31523 STDIN

31524 The *qsub* utility reads the script of the batch job from standard input if the script operand is  
 31525 omitted or is the single character ' - '.

### 31526 INPUT FILES

31527 In addition to binding the file indicated by the *script* operand to the batch job, the *qsub* utility  
 31528 reads the script file and acts on directives in the script.

### 31529 ENVIRONMENT VARIABLES

31530 The following environment variables shall affect the execution of *qsub*:

31531 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 31532 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 31533 Internationalization Variables for the precedence of internationalization variables  
 31534 used to determine the values of locale categories.)

31535 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 31536 internationalization variables.

31537 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 31538 characters (for example, single-byte as opposed to multi-byte characters in  
 31539 arguments).

31540 *LC\_MESSAGES*

31541 Determine the locale that should be used to affect the format and contents of  
 31542 diagnostic messages written to standard error.

31543 *LOGNAME* Determine the login name of the user.

31544 *PBS\_DPREFIX*

31545 Determine the default prefix for directives within the script.

31546 *SHELL* Determine the pathname of the preferred command language interpreter of the  
 31547 user.

31548 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is  
 31549 unset or null, an unspecified default timezone shall be used.

### 31550 ASYNCHRONOUS EVENTS

31551 Once created, a batch job exists until it exits, aborts, or is deleted.

31552 After a batch job is created by the *qsub* utility, batch servers might route, execute, modify, or  
 31553 delete the batch job.

### 31554 STDOUT

31555 The *qsub* utility writes the batch *job\_identifier* assigned to the batch job to standard output, unless  
 31556 the **-z** option is specified.

31557 **STDERR**

31558           The standard error shall be used only for diagnostic messages.

31559 **OUTPUT FILES**

31560           None.

31561 **EXTENDED DESCRIPTION**31562           **Script Preservation**

31563           The *qsub* utility shall make the script available to the server executing the batch job in such a way  
31564           that the server executes the script as it exists at the time of submission.

31565           The *qsub* utility can send a copy of the script to the server with the *Queue Job Request* or store a  
31566           temporary copy of the script in a location specified to the server.

31567           **Option Specification**

31568           A script can contain directives to the *qsub* utility.

31569           The *qsub* utility shall scan the lines of the script for directives, skipping blank lines, until the first  
31570           line that begins with a string other than the directive string; if directives occur on subsequent  
31571           lines, the utility shall ignore those directives.

31572           Lines are separated by a <newline>. If the first line of the script begins with "#!" or a colon  
31573           (' : '), then it is skipped. The *qsub* utility shall process a line in the script as a directive if and  
31574           only if the string of characters from the first non-white-space character on the line until the first  
31575           <space> or <tab> on the line match the directive prefix. If a line in the script contains a directive  
31576           and the final characters of the line are backslash (' \ ') and <newline>, then the next line shall be  
31577           interpreted as a continuation of that directive.

31578           The *qsub* utility shall process the options and option-arguments contained on the directive prefix  
31579           line using the same syntax as if the options were input on the *qsub* utility.

31580           The *qsub* utility shall continue to process a directive prefix line until after a <newline> is  
31581           encountered. An implementation may ignore lines which, according to the syntax of the shell  
31582           that will interpret the script, are comments. An implementation shall describe in the  
31583           conformance document the format of any shell comments that it will recognize.

31584           If an option is present in both a directive and the arguments to the *qsub* utility, the utility shall  
31585           ignore the option and the corresponding option-argument, if any, in the directive.

31586           If an option that is present in the directive is not present in the arguments to the *qsub* utility, the  
31587           utility shall process the option and the option-argument, if any.

31588           In order of preference, the *qsub* utility shall select the directive prefix from one of the following  
31589           sources:

- 31590           • If the **-C** option is presented to the utility, the value of the *directive\_prefix* option-argument
- 31591           • If the environment variable *PBS\_DPREFIX* is defined, the value of that variable
- 31592           • The four-character string "#PBS" encoded in the portable character set

31593           If the **-C** option is present in the script file it shall be ignored.

31594 **EXIT STATUS**

31595           The following exit values shall be returned:

31596           0   Successful completion.



31597 >0 An error occurred.

31598 **CONSEQUENCES OF ERRORS**

31599 Default.

31600 **APPLICATION USAGE**

31601 None.

31602 **EXAMPLES**

31603 None.

31604 **RATIONALE**

31605 The *qsub* utility allows users to create a batch job that will process the script specified as the  
31606 operand of the utility.

31607 The options of the *qsub* utility allow users to control many aspects of the queuing and execution  
31608 of a batch job.

31609 The **-a** option allows users to designate the time after which the batch job will become eligible to  
31610 run. By specifying an execution time, users can take advantage of resources at off-peak hours,  
31611 synchronize jobs with chronologically predictable events, and perhaps take advantage of off-  
31612 peak pricing of computing time. For these reasons and others, a timing option is existing practice  
31613 on the part of almost every batch system, including NQS.

31614 The **-A** option allows users to specify the account that will be charged for the batch job. Support  
31615 for account is not mandatory for conforming batch servers.

31616 The **-C** option allows users to prescribe the prefix for directives within the script file. The default  
31617 prefix "#PBS" may be inappropriate if the script will be interpreted with an alternate shell, as  
31618 specified by the **-S** option.

31619 The **-c** option allows users to establish the checkpointing interval for their jobs. A checkpointing  
31620 system, which is not defined by this volume of IEEE Std 1003.1-2001, allows recovery of a batch  
31621 job at the most recent checkpoint in the event of a crash. Checkpointing is typically used for jobs  
31622 that consume expensive computing time or must meet a critical schedule. Users should be  
31623 allowed to make the tradeoff between the overhead of checkpointing and the risk to the timely  
31624 completion of the batch job; therefore, this volume of IEEE Std 1003.1-2001 provides the  
31625 checkpointing interval option. Support for checkpointing is optional for batch servers.

31626 The **-e** option allows users to redirect the standard error streams of their jobs to a non-default  
31627 path. For example, if the submitted script generally produces a great deal of useless error output,  
31628 a user might redirect the standard error output to the null device. Or, if the file system holding  
31629 the default location (the home directory of the user) has too little free space, the user might  
31630 redirect the standard error stream to a file in another file system.

31631 The **-h** option allows users to create a batch job that is held until explicitly released. The ability  
31632 to create a held job is useful when some external event must complete before the batch job can  
31633 execute. For example, the user might submit a held job and release it when the system load has  
31634 dropped.

31635 The **-j** option allows users to merge the standard error of a batch job into its standard output  
31636 stream, which has the advantage of showing the sequential relationship between output and  
31637 error messages.

31638 The **-m** option allows users to designate those points in the execution of a batch job at which  
31639 mail will be sent to the submitting user, or to the account(s) indicated by the **-M** option. By  
31640 requesting mail notification at points of interest in the life of a job, the submitting user, or other  
31641 designated users, can track the progress of a batch job.

31642 The **-N** option allows users to associate a name with the batch job. The job name in no way  
31643 affects the processing of the batch job, but rather serves as a mnemonic handle for users. For  
31644 example, the batch job name can help the user distinguish between multiple jobs listed by the  
31645 *qstat* utility.

31646 The **-o** option allows users to redirect the standard output stream. A user might, for example,  
31647 wish to redirect to the null device the standard output stream of a job that produces copious yet  
31648 superfluous output.

31649 The **-P** option allows users to designate the relative priority of a batch job for selection from a  
31650 queue.

31651 The **-q** option allows users to specify an initial queue for the batch job. If the user specifies a  
31652 routing queue, the batch server routes the batch job to another queue for execution or further  
31653 routing. If the user specifies a non-routing queue, the batch server of the queue eventually  
31654 executes the batch job.

31655 The **-r** option allows users to control whether the submitted job will be rerun if the controlling  
31656 batch node fails during execution of the batch job. The **-r** option likewise allows users to  
31657 indicate whether or not the batch job is eligible to be rerun by the *qrerun* utility. Some jobs cannot  
31658 be correctly rerun because of changes they make in the state of databases or other aspects of  
31659 their environment. This volume of IEEE Std 1003.1-2001 specifies that the default, if the **-r**  
31660 option is not presented to the utility, will be that the batch job cannot be rerun, since the result of  
31661 rerunning a non-runnable job might be catastrophic.

31662 The **-S** option allows users to specify the program (usually a shell) that will be invoked to  
31663 process the script of the batch job. This option has been modified to allow a list of shell names  
31664 and locations associated with different hosts.

31665 The **-u** option is useful when the submitting user is authorized to use more than one account on  
31666 a given host, in which case the **-u** option allows the user to select from among those accounts.  
31667 The option-argument is a list of user-host pairs, so that the submitting user can provide different  
31668 user identifiers for different nodes in the event the batch job is routed. The **-u** option provides a  
31669 lot of flexibility to accommodate sites with complex account structures. Users that have the  
31670 same user identifier on all the hosts they are authorized to use will not need to use the **-u** option.

31671 The **-V** option allows users to export all their current environment variables, as of the time the  
31672 batch job is submitted, to the context of the processes of the batch job.

31673 The **-v** option allows users to export specific environment variables from their current process  
31674 to the processes of the batch job.

31675 The **-z** option allows users to suppress the writing of the batch job identifier to standard output.  
31676 The **-z** option is an existing NQS practice that has been standardized.

31677 Historically, the *qsub* utility has served the batch job-submission function in the NQS system, the  
31678 existing practice on which it is based. Some changes and additions have been made to the *qsub*  
31679 utility in this volume of IEEE Std 1003.1-2001, *vis-a-vis* NQS, as a result of the growing pool of  
31680 experience with distributed batch systems.

31681 The set of features of the *qsub* utility as defined in this volume of IEEE Std 1003.1-2001 appears to  
31682 incorporate all the common existing practice on potentially conforming platforms.

#### 31683 **FUTURE DIRECTIONS**

31684 None.

31685 **SEE ALSO**31686 Chapter 3 (on page 101), *qrerun*, *qstat*, *touch*31687 **CHANGE HISTORY**

31688 Derived from IEEE Std 1003.2d-1994.

31689 **Issue 6**31690 The **-I** option has been removed as there is no portable description of the resources that are  
31691 allowed or required by the batch job.

31692 **NAME**

31693           read — read a line from standard input

31694 **SYNOPSIS**

31695           read [-r] var...

31696 **DESCRIPTION**31697           The *read* utility shall read a single line from standard input.

31698           By default, unless the **-r** option is specified, backslash ('\**\**') shall act as an escape character, as  
 31699           described in Section 2.2.1 (on page 30). If standard input is a terminal device and the invoking  
 31700           shell is interactive, *read* shall prompt for a continuation line when:

- 31701           • The shell reads an input line ending with a backslash, unless the **-r** option is specified.
- 31702           • A here-document is not terminated after a <newline> is entered.

31703           The line shall be split into fields as in the shell (see Section 2.6.5 (on page 42)); the first field shall  
 31704           be assigned to the first variable *var*, the second field to the second variable *var*, and so on. If  
 31705           there are fewer *var* operands specified than there are fields, the leftover fields and their  
 31706           intervening separators shall be assigned to the last *var*. If there are fewer fields than *vars*, the  
 31707           remaining *vars* shall be set to empty strings.

31708           The setting of variables specified by the *var* operands shall affect the current shell execution  
 31709           environment; see Section 2.12 (on page 61). If it is called in a subshell or separate utility  
 31710           execution environment, such as one of the following:

```
31711 (read foo)
31712 nohup read ...
31713 find . -exec read ... \;
```

31714           it shall not affect the shell variables in the caller's environment.

31715 **OPTIONS**

31716           The *read* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 31717           12.2, Utility Syntax Guidelines.

31718           The following option is supported:

31719           **-r**           Do not treat a backslash character in any special way. Consider each backslash to  
 31720           be part of the input line.

31721 **OPERANDS**

31722           The following operand shall be supported:

31723           *var*           The name of an existing or nonexisting shell variable.

31724 **STDIN**

31725           The standard input shall be a text file.

31726 **INPUT FILES**

31727           None.

31728 **ENVIRONMENT VARIABLES**31729           The following environment variables shall affect the execution of *read*:

31730           **IFS**           Determine the internal field separators used to delimit fields; see Section 2.5.3 (on  
 31731           page 34).

31732           **LANG**          Provide a default value for the internationalization variables that are unset or null.  
 31733           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 31734           Internationalization Variables for the precedence of internationalization variables)

31735 used to determine the values of locale categories.)

31736 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
31737 internationalization variables.

31738 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
31739 characters (for example, single-byte as opposed to multi-byte characters in  
31740 arguments).

31741 **LC\_MESSAGES**  
31742 Determine the locale that should be used to affect the format and contents of  
31743 diagnostic messages written to standard error.

31744 **NSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

31745 **PS2** Provide the prompt string that an interactive shell shall write to standard error  
31746 when a line ending with a backslash is read and the **-r** option was not specified, or  
31747 if a here-document is not terminated after a <newline> is entered.

31748 **ASYNCHRONOUS EVENTS**  
31749 Default.

31750 **STDOUT**  
31751 Not used.

31752 **STDERR**  
31753 The standard error shall be used for diagnostic messages and prompts for continued input.

31754 **OUTPUT FILES**  
31755 None.

31756 **EXTENDED DESCRIPTION**  
31757 None.

31758 **EXIT STATUS**  
31759 The following exit values shall be returned:  
31760 0 Successful completion.  
31761 >0 End-of-file was detected or an error occurred.

31762 **CONSEQUENCES OF ERRORS**  
31763 Default.

31764 **APPLICATION USAGE**  
31765 The **-r** option is included to enable *read* to subsume the purpose of the *line* utility, which is not  
31766 included in IEEE Std 1003.1-2001.  
31767 The results are undefined if an end-of-file is detected following a backslash at the end of a line  
31768 when **-r** is not specified.

31769 **EXAMPLES**  
31770 The following command:  
31771 while read -r xx yy  
31772 do  
31773 printf "%s %s\n" "\$yy" "\$xx"  
31774 done < *input\_file*  
31775 prints a file with the first field of each line moved to the end of the line.

31776 **RATIONALE**

31777       The *read* utility historically has been a shell built-in. It was separated off into its own utility to  
31778       take advantage of the richer description of functionality introduced by this volume of  
31779       IEEE Std 1003.1-2001.

31780       Since *read* affects the current shell execution environment, it is generally provided as a shell  
31781       regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
31782       of the following:

31783        (read foo)  
31784        nohup read ...  
31785        find . -exec read ... \;

31786       it does not affect the shell variables in the environment of the caller.

31787 **FUTURE DIRECTIONS**

31788       None.

31789 **SEE ALSO**

31790       Chapter 2 (on page 29)

31791 **CHANGE HISTORY**

31792       First released in Issue 2.

31793 **NAME**

31794           renice — set nice values of running processes

31795 **SYNOPSIS**31796 UP        renice -n *increment* [-g | -p | -u] *ID* ...

31797

31798 **DESCRIPTION**

31799       The *renice* utility shall request that the nice values (see the Base Definitions volume of  
 31800 IEEE Std 1003.1-2001, Section 3.239, Nice Value) of one or more running processes be changed.  
 31801 By default, the applicable processes are specified by their process IDs. When a process group is  
 31802 specified (see -g), the request shall apply to all processes in the process group.

31803       The nice value shall be bounded in an implementation-defined manner. If the requested  
 31804 *increment* would raise or lower the nice value of the executed utility beyond implementation-  
 31805 defined limits, then the limit whose value was exceeded shall be used.

31806       When a user is *reniced*, the request applies to all processes whose saved set-user-ID matches the  
 31807 user ID corresponding to the user.

31808       Regardless of which options are supplied or any other factor, *renice* shall not alter the nice values  
 31809 of any process unless the user requesting such a change has appropriate privileges to do so for  
 31810 the specified process. If the user lacks appropriate privileges to perform the requested action, the  
 31811 utility shall return an error status.

31812       The saved set-user-ID of the user's process shall be checked instead of its effective user ID when  
 31813 *renice* attempts to determine the user ID of the process in order to determine whether the user  
 31814 has appropriate privileges.

31815 **OPTIONS**

31816       The *renice* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 31817 12.2, Utility Syntax Guidelines.

31818       The following options shall be supported:

31819       -g           Interpret all operands as unsigned decimal integer process group IDs.

31820       -n *increment* Specify how the nice value of the specified process or processes is to be adjusted.  
 31821       The *increment* option-argument is a positive or negative decimal integer that shall  
 31822       be used to modify the nice value of the specified process or processes.

31823       Positive *increment* values shall cause a lower nice value. Negative *increment* values  
 31824       may require appropriate privileges and shall cause a higher nice value.

31825       -p           Interpret all operands as unsigned decimal integer process IDs. The -p option is  
 31826       the default if no options are specified.

31827       -u           Interpret all operands as users. If a user exists with a user name equal to the  
 31828       operand, then the user ID of that user is used in further processing. Otherwise, if  
 31829       the operand represents an unsigned decimal integer, it shall be used as the numeric  
 31830       user ID of the user.

31831 **OPERANDS**

31832       The following operands shall be supported:

31833       *ID*           A process ID, process group ID, or user name/user ID, depending on the option  
 31834       selected.

31835 **STDIN**

31836 Not used.

31837 **INPUT FILES**

31838 None.

31839 **ENVIRONMENT VARIABLES**31840 The following environment variables shall affect the execution of *renice*:

31841 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 31842 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 31843 Internationalization Variables for the precedence of internationalization variables  
 31844 used to determine the values of locale categories.)

31845 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 31846 internationalization variables.

31847 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 31848 characters (for example, single-byte as opposed to multi-byte characters in  
 31849 arguments).

31850 *LC\_MESSAGES*

31851 Determine the locale that should be used to affect the format and contents of  
 31852 diagnostic messages written to standard error.

31853 *NSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

31854 **ASYNCHRONOUS EVENTS**

31855 Default.

31856 **STDOUT**

31857 Not used.

31858 **STDERR**

31859 The standard error shall be used only for diagnostic messages.

31860 **OUTPUT FILES**

31861 None.

31862 **EXTENDED DESCRIPTION**

31863 None.

31864 **EXIT STATUS**

31865 The following exit values shall be returned:

31866 0 Successful completion.

31867 &gt;0 An error occurred.

31868 **CONSEQUENCES OF ERRORS**

31869 Default.



31870 **APPLICATION USAGE**

31871       None.

31872 **EXAMPLES**

31873       1. Adjust the nice value so that process IDs 987 and 32 would have a lower nice value:

31874       `renice -n 5 -p 987 32`31875       2. Adjust the nice value so that group IDs 324 and 76 would have a higher nice value, if the  
31876       user has the appropriate privileges to do so:31877       `renice -n -4 -g 324 76`31878       3. Adjust the nice value so that numeric user ID 8 and user **sas** would have a lower nice  
31879       value:31880       `renice -n 4 -u 8 sas`31881       Useful nice value increments on historical systems include 19 or 20 (the affected processes run  
31882       only when nothing else in the system attempts to run) and any negative number (to make  
31883       processes run faster).31884 **RATIONALE**31885       The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-  
31886       argument. However, for clarity, they have been included in the OPTIONS section, rather than  
31887       the OPERANDS section.31888       The definition of nice value is not intended to suggest that all processes in a system have  
31889       priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the  
31890       System Interfaces volume of IEEE Std 1003.1-2001 make the notion of a single underlying  
31891       priority for all scheduling policies problematic. Some implementations may implement the *nice*-  
31892       related features to affect all processes on the system, others to affect just the general time-  
31893       sharing activities implied by this volume of IEEE Std 1003.1-2001, and others may have no effect  
31894       at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of  
31895       implementation strategies are possible.31896       Originally, this utility was written in the historical manner, using the term “nice value”. This  
31897       was always a point of concern with users because it was never intuitively obvious what this  
31898       meant. With a newer version of *renice*, which used the term “system scheduling priority”, it was  
31899       hoped that novice users could better understand what this utility was meant to do. Also, it  
31900       would be easier to document what the utility was meant to do. Unfortunately, the addition of  
31901       the POSIX realtime scheduling capabilities introduced the concepts of process and thread  
31902       scheduling priorities that were totally unaffected by the *nice/renice* utilities or the  
31903       *nice()/setpriority()* functions. Continuing to use the term “system scheduling priority” would  
31904       have incorrectly suggested that these utilities and functions were indeed affecting these realtime  
31905       priorities. It was decided to revert to the historical term “nice value” to reference this unrelated  
31906       process attribute.31907       Although this utility has use by system administrators (and in fact appears in the system  
31908       administration portion of the BSD documentation), the standard developers considered that it  
31909       was very useful for individual end users to control their own processes.31910 **FUTURE DIRECTIONS**

31911       None.

31912 **SEE ALSO**31913 *nice*31914 **CHANGE HISTORY**

31915 First released in Issue 4.

31916 **Issue 5**31917 In the SYNOPSIS, an ellipsis is added to the **-u** option in all three obsolescent forms.31918 **Issue 6**

31919 This utility is marked as part of the User Portability Utilities option.

31920 The APPLICATION USAGE section is added.

31921 The obsolescent forms of the SYNOPSIS are removed.

31922 Text previously conditional on POSIX\_SAVED\_IDS is mandatory in this issue. This is a FIPS  
31923 requirement.

31924 **NAME**

31925           rm — remove directory entries

31926 **SYNOPSIS**31927           rm [-fiRr] *file...*31928 **DESCRIPTION**31929           The *rm* utility shall remove the directory entry specified by each *file* argument.

31930           If either of the files dot or dot-dot are specified as the basename portion of an operand (that is,  
 31931           the final pathname component), *rm* shall write a diagnostic message to standard error and do  
 31932           nothing more with such operands.

31933           For each *file* the following steps shall be taken:

- 31934           1. If the *file* does not exist:
    - 31935               a. If the **-f** option is not specified, *rm* shall write a diagnostic message to standard error.
    - 31936               b. Go on to any remaining *files*.
  - 31937           2. If *file* is of type directory, the following steps shall be taken:
    - 31938               a. If neither the **-R** option nor the **-r** option is specified, *rm* shall write a diagnostic  
 31939               message to standard error, do nothing more with *file*, and go on to any remaining  
 31940               files.
    - 31941               b. If the **-f** option is not specified, and either the permissions of *file* do not permit  
 31942               writing and the standard input is a terminal or the **-i** option is specified, *rm* shall  
 31943               write a prompt to standard error and read a line from the standard input. If the  
 31944               response is not affirmative, *rm* shall do nothing more with the current file and go on  
 31945               to any remaining files.
    - 31946               c. For each entry contained in *file*, other than dot or dot-dot, the four steps listed here (1  
 31947               to 4) shall be taken with the entry as if it were a *file* operand. The *rm* utility shall not  
 31948               traverse directories by following symbolic links into other parts of the hierarchy, but  
 31949               shall remove the links themselves.
    - 31950               d. If the **-i** option is specified, *rm* shall write a prompt to standard error and read a line  
 31951               from the standard input. If the response is not affirmative, *rm* shall do nothing more  
 31952               with the current file, and go on to any remaining files.
  - 31953           3. If *file* is not of type directory, the **-f** option is not specified, and either the permissions of  
 31954           *file* do not permit writing and the standard input is a terminal or the **-i** option is specified,  
 31955           *rm* shall write a prompt to the standard error and read a line from the standard input. If the  
 31956           response is not affirmative, *rm* shall do nothing more with the current file and go on to any  
 31957           remaining files.
  - 31958           4. If the current file is a directory, *rm* shall perform actions equivalent to the *rmdir()* function  
 31959           defined in the System Interfaces volume of IEEE Std 1003.1-2001 called with a pathname of  
 31960           the current file used as the *path* argument. If the current file is not a directory, *rm* shall  
 31961           perform actions equivalent to the *unlink()* function defined in the System Interfaces  
 31962           volume of IEEE Std 1003.1-2001 called with a pathname of the current file used as the *path*  
 31963           argument.
- 31964           If this fails for any reason, *rm* shall write a diagnostic message to standard error, do  
 31965           nothing more with the current file, and go on to any remaining files.
- 31966           The *rm* utility shall be able to descend to arbitrary depths in a file hierarchy, and shall not fail  
 31967           due to path length limitations (unless an operand specified by the user exceeds system

31968 limitations).

### 31969 OPTIONS

31970 The *rm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
31971 Utility Syntax Guidelines.

31972 The following options shall be supported:

31973 **-f** Do not prompt for confirmation. Do not write diagnostic messages or modify the  
31974 exit status in the case of nonexistent operands. Any previous occurrences of the **-i**  
31975 option shall be ignored.

31976 **-i** Prompt for confirmation as described previously. Any previous occurrences of the  
31977 **-f** option shall be ignored.

31978 **-R** Remove file hierarchies. See the DESCRIPTION.

31979 **-r** Equivalent to **-R**.

### 31980 OPERANDS

31981 The following operand shall be supported:

31982 *file* A pathname of a directory entry to be removed.

### 31983 STDIN

31984 The standard input shall be used to read an input line in response to each prompt specified in  
31985 the STDOUT section. Otherwise, the standard input shall not be used.

### 31986 INPUT FILES

31987 None.

### 31988 ENVIRONMENT VARIABLES

31989 The following environment variables shall affect the execution of *rm*:

31990 **LANG** Provide a default value for the internationalization variables that are unset or null.  
31991 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
31992 Internationalization Variables for the precedence of internationalization variables  
31993 used to determine the values of locale categories.)

31994 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
31995 internationalization variables.

31996 **LC\_COLLATE**  
31997 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
31998 character collating elements used in the extended regular expression defined for  
31999 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

32000 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
32001 characters (for example, single-byte as opposed to multi-byte characters in  
32002 arguments) and the behavior of character classes within regular expressions used  
32003 in the extended regular expression defined for the **yesexpr** locale keyword in the  
32004 *LC\_MESSAGES* category.

32005 **LC\_MESSAGES**  
32006 Determine the locale for the processing of affirmative responses that should be  
32007 used to affect the format and contents of diagnostic messages written to standard  
32008 error.

32009 **NSI** **NLS\_PATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

32010 **ASYNCHRONOUS EVENTS**

32011 Default.

32012 **STDOUT**

32013 Not used.

32014 **STDERR**

32015 Prompts shall be written to standard error under the conditions specified in the DESCRIPTION  
 32016 and OPTIONS sections. The prompts shall contain the *file* pathname, but their format is  
 32017 otherwise unspecified. The standard error also shall be used for diagnostic messages.

32018 **OUTPUT FILES**

32019 None.

32020 **EXTENDED DESCRIPTION**

32021 None.

32022 **EXIT STATUS**

32023 The following exit values shall be returned:

32024 0 All of the named directory entries for which *rm* performed actions equivalent to the *rmdir()*  
 32025 or *unlink()* functions were removed.

32026 &gt;0 An error occurred.

32027 **CONSEQUENCES OF ERRORS**

32028 Default.

32029 **APPLICATION USAGE**

32030 The *rm* utility is forbidden to remove the names dot and dot-dot in order to avoid the  
 32031 consequences of inadvertently doing something like:

32032 `rm -r .*`

32033 Some implementations do not permit the removal of the last link to an executable binary file that  
 32034 is being executed; see the [EBUSY] error in the *unlink()* function defined in the System Interfaces  
 32035 volume of IEEE Std 1003.1-2001. Thus, the *rm* utility can fail to remove such files.

32036 The *-i* option causes *rm* to prompt and read the standard input even if the standard input is not  
 32037 a terminal, but in the absence of *-i* the mode prompting is not done when the standard input is  
 32038 not a terminal.

32039 **EXAMPLES**

32040 1. The following command:

32041 `rm a.out core`32042 removes the directory entries: **a.out** and **core**.

32043 2. The following command:

32044 `rm -Rf junk`32045 removes the directory **junk** and all its contents, without prompting.32046 **RATIONALE**

32047 For absolute clarity, paragraphs (2b) and (3) in the DESCRIPTION of *rm* describing the behavior  
 32048 when prompting for confirmation, should be interpreted in the following manner:

32049 `if ((NOT f_option) AND`  
 32050 `((not_writable AND input_is_terminal) OR i_option))`

32051 The exact format of the interactive prompts is unspecified. Only the general nature of the  
32052 contents of prompts are specified because implementations may desire more descriptive  
32053 prompts than those used on historical implementations. Therefore, an application not using the  
32054 `-f` option, or using the `-i` option, relies on the system to provide the most suitable dialog directly  
32055 with the user, based on the behavior specified.

32056 The `-r` option is historical practice on all known systems. The synonym `-R` option is provided  
32057 for consistency with the other utilities in this volume of IEEE Std 1003.1-2001 that provide  
32058 options requesting recursive descent through the file hierarchy.

32059 The behavior of the `-f` option in historical versions of *rm* is inconsistent. In general, along with  
32060 “forcing” the unlink without prompting for permission, it always causes diagnostic messages to  
32061 be suppressed and the exit status to be unmodified for nonexistent operands and files that  
32062 cannot be unlinked. In some versions, however, the `-f` option suppresses usage messages and  
32063 system errors as well. Suppressing such messages is not a service to either shell scripts or users.

32064 It is less clear that error messages regarding files that cannot be unlinked (removed) should be  
32065 suppressed. Although this is historical practice, this volume of IEEE Std 1003.1-2001 does not  
32066 permit the `-f` option to suppress such messages.

32067 When given the `-r` and `-i` options, historical versions of *rm* prompt the user twice for each  
32068 directory, once before removing its contents and once before actually attempting to delete the  
32069 directory entry that names it. This allows the user to “prune” the file hierarchy walk. Historical  
32070 versions of *rm* were inconsistent in that some did not do the former prompt for directories  
32071 named on the command line and others had obscure prompting behavior when the `-i` option  
32072 was specified and the permissions of the file did not permit writing. The POSIX Shell and  
32073 Utilities *rm* differs little from historic practice, but does require that prompts be consistent.  
32074 Historical versions of *rm* were also inconsistent in that prompts were done to both standard  
32075 output and standard error. This volume of IEEE Std 1003.1-2001 requires that prompts be done  
32076 to standard error, for consistency with *cp* and *mv*, and to allow historical extensions to *rm* that  
32077 provide an option to list deleted files on standard output.

32078 The *rm* utility is required to descend to arbitrary depths so that any file hierarchy may be  
32079 deleted. This means, for example, that the *rm* utility cannot run out of file descriptors during its  
32080 descent (that is, if the number of file descriptors is limited, *rm* cannot be implemented in the  
32081 historical fashion where one file descriptor is used per directory level). Also, *rm* is not permitted  
32082 to fail because of path length restrictions, unless an operand specified by the user is longer than  
32083 {PATH\_MAX}.

32084 The *rm* utility removes symbolic links themselves, not the files they refer to, as a consequence of  
32085 the dependence on the *unlink()* functionality, per the DESCRIPTION. When removing  
32086 hierarchies with `-r` or `-R`, the prohibition on following symbolic links has to be made explicit.

#### 32087 FUTURE DIRECTIONS

32088 None.

#### 32089 SEE ALSO

32090 *rmdir*, the System Interfaces volume of IEEE Std 1003.1-2001, *remove()*, *rmdir()*, *unlink()*

#### 32091 CHANGE HISTORY

32092 First released in Issue 2.

#### 32093 Issue 5

32094 The FUTURE DIRECTIONS section is added.

32095 **Issue 6**

32096 Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft  
32097 standard.

32098 **NAME**32099           rm~~del~~ — remove a delta from an SCCS file (**DEVELOPMENT**)32100 **SYNOPSIS**32101 xSI       rm~~del~~ -r *SID file...*

32102

32103 **DESCRIPTION**

32104       The *rm~~del~~* utility shall remove the delta specified by the *SID* from each named SCCS file. The  
 32105       delta to be removed shall be the most recent delta in its branch in the delta chain of each named  
 32106       SCCS file. In addition, the application shall ensure that the *SID* specified is not that of a version  
 32107       being edited for the purpose of making a delta; that is, if a *p-file* (see *get*) exists for the named  
 32108       SCCS file, the *SID* specified shall not appear in any entry of the *p-file*.

32109       Removal of a delta shall be restricted to:

- 32110           1. The user who made the delta
- 32111           2. The owner of the SCCS file
- 32112           3. The owner of the directory containing the SCCS file

32113 **OPTIONS**

32114       The *rm~~del~~* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 32115       12.2, Utility Syntax Guidelines.

32116       The following option shall be supported:

32117       -r *SID*       Specify the SCCS identification string (*SID*) of the delta to be deleted.32118 **OPERANDS**

32119       The following operand shall be supported:

32120       *file*           A pathname of an existing SCCS file or a directory. If *file* is a directory, the *rm~~del~~*  
 32121       utility shall behave as though each file in the directory were specified as a named  
 32122       file, except that non-SCCS files (last component of the pathname does not begin  
 32123       with **s**.) and unreadable files shall be silently ignored.

32124                       If exactly one *file* operand appears, and it is '-', the standard input shall be read;  
 32125       each line of the standard input is taken to be the name of an SCCS file to be  
 32126       processed. Non-SCCS files and unreadable files shall be silently ignored.

32127 **STDIN**

32128       The standard input shall be a text file used only when the *file* operand is specified as '-'. Each  
 32129       line of the text file shall be interpreted as an SCCS pathname.

32130 **INPUT FILES**

32131       The SCCS files shall be files of unspecified format.

32132 **ENVIRONMENT VARIABLES**32133       The following environment variables shall affect the execution of *rm~~del~~*:

32134       *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 32135       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 32136       Internationalization Variables for the precedence of internationalization variables  
 32137       used to determine the values of locale categories.)

32138       *LC\_ALL*          If set to a non-empty string value, override the values of all the other  
 32139       internationalization variables.



|       |                               |                                                                                                                                                                                                                                                                     |
|-------|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 32140 | <b>LC_CTYPE</b>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                           |
| 32141 |                               |                                                                                                                                                                                                                                                                     |
| 32142 |                               |                                                                                                                                                                                                                                                                     |
| 32143 | <b>LC_MESSAGES</b>            |                                                                                                                                                                                                                                                                     |
| 32144 |                               | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                        |
| 32145 |                               |                                                                                                                                                                                                                                                                     |
| 32146 | <b>NLSPATH</b>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                               |
| 32147 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                     |
| 32148 |                               | Default.                                                                                                                                                                                                                                                            |
| 32149 | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                     |
| 32150 |                               | Not used.                                                                                                                                                                                                                                                           |
| 32151 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                     |
| 32152 |                               | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                      |
| 32153 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                     |
| 32154 |                               | The SCCS files shall be files of unspecified format. During processing of a <i>file</i> , a temporary <i>x-file</i> , as described in <i>admin</i> , may be created and deleted; a locking <i>z-file</i> , as described in <i>get</i> , may be created and deleted. |
| 32155 |                               |                                                                                                                                                                                                                                                                     |
| 32156 |                               |                                                                                                                                                                                                                                                                     |
| 32157 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                     |
| 32158 |                               | None.                                                                                                                                                                                                                                                               |
| 32159 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                     |
| 32160 |                               | The following exit values shall be returned:                                                                                                                                                                                                                        |
| 32161 | 0                             | Successful completion.                                                                                                                                                                                                                                              |
| 32162 | >0                            | An error occurred.                                                                                                                                                                                                                                                  |
| 32163 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                     |
| 32164 |                               | Default.                                                                                                                                                                                                                                                            |
| 32165 | <b>APPLICATION USAGE</b>      |                                                                                                                                                                                                                                                                     |
| 32166 |                               | None.                                                                                                                                                                                                                                                               |
| 32167 | <b>EXAMPLES</b>               |                                                                                                                                                                                                                                                                     |
| 32168 |                               | None.                                                                                                                                                                                                                                                               |
| 32169 | <b>RATIONALE</b>              |                                                                                                                                                                                                                                                                     |
| 32170 |                               | None.                                                                                                                                                                                                                                                               |
| 32171 | <b>FUTURE DIRECTIONS</b>      |                                                                                                                                                                                                                                                                     |
| 32172 |                               | None.                                                                                                                                                                                                                                                               |
| 32173 | <b>SEE ALSO</b>               |                                                                                                                                                                                                                                                                     |
| 32174 |                               | <i>admin</i> , <i>delta</i> , <i>get</i> , <i>prs</i>                                                                                                                                                                                                               |
| 32175 | <b>CHANGE HISTORY</b>         |                                                                                                                                                                                                                                                                     |
| 32176 |                               | First released in Issue 2.                                                                                                                                                                                                                                          |
| 32177 | <b>Issue 6</b>                |                                                                                                                                                                                                                                                                     |
| 32178 |                               | The normative text is reworded to avoid use of the term “must” for application requirements.                                                                                                                                                                        |

32179 **NAME**

32180           rmdir — remove directories

32181 **SYNOPSIS**32182           rmdir [-p] *dir*...32183 **DESCRIPTION**32184           The *rmdir* utility shall remove the directory entry specified by each *dir* operand.32185           For each *dir* operand, the *rmdir* utility shall perform actions equivalent to the *rmdir()* function  
32186           called with the *dir* operand as its only argument.32187           Directories shall be processed in the order specified. If a directory and a subdirectory of that  
32188           directory are specified in a single invocation of the *rmdir* utility, the application shall specify the  
32189           subdirectory before the parent directory so that the parent directory will be empty when the  
32190           *rmdir* utility tries to remove it.32191 **OPTIONS**32192           The *rmdir* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
32193           12.2, Utility Syntax Guidelines.

32194           The following option shall be supported:

32195           **-p**           Remove all directories in a pathname. For each *dir* operand:

- 32196                           1. The directory entry it names shall be removed.
- 
- 32197                           2. If the
- dir*
- operand includes more than one pathname component, effects
- 
- 32198                           equivalent to the following command shall occur:

32199                                 rmdir -p \$(dirname *dir*)32200 **OPERANDS**

32201           The following operand shall be supported:

32202           *dir*           A pathname of an empty directory to be removed.32203 **STDIN**

32204           Not used.

32205 **INPUT FILES**

32206           None.

32207 **ENVIRONMENT VARIABLES**32208           The following environment variables shall affect the execution of *rmdir*:32209           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
32210                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
32211                           Internationalization Variables for the precedence of internationalization variables  
32212                           used to determine the values of locale categories.)32213           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
32214                           internationalization variables.32215           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
32216                           characters (for example, single-byte as opposed to multi-byte characters in  
32217                           arguments).32218           **LC\_MESSAGES**32219                           Determine the locale that should be used to affect the format and contents of  
32220                           diagnostic messages written to standard error.

32221 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 32222 ASYNCHRONOUS EVENTS

32223 Default.

## 32224 STDOUT

32225 Not used.

## 32226 STDERR

32227 The standard error shall be used only for diagnostic messages.

## 32228 OUTPUT FILES

32229 None.

## 32230 EXTENDED DESCRIPTION

32231 None.

## 32232 EXIT STATUS

32233 The following exit values shall be returned:

32234 0 Each directory entry specified by a *dir* operand was removed successfully.

32235 >0 An error occurred.

## 32236 CONSEQUENCES OF ERRORS

32237 Default.

## 32238 APPLICATION USAGE

32239 The definition of an empty directory is one that contains, at most, directory entries for dot and dot-dot.

## 32241 EXAMPLES

32242 If a directory **a** in the current directory is empty except it contains a directory **b** and **a/b** is empty  
32243 except it contains a directory **c**:

32244 `rmdir -p a/b/c`

32245 removes all three directories.

## 32246 RATIONALE

32247 On historical System V systems, the **-p** option also caused a message to be written to the  
32248 standard output. The message indicated whether the whole path was removed or whether part  
32249 of the path remained for some reason. The STDERR section requires this diagnostic when the  
32250 entire path specified by a *dir* operand is not removed, but does not allow the status message  
32251 reporting success to be written as a diagnostic.

32252 The *rmdir* utility on System V also included a **-s** option that suppressed the informational  
32253 message output by the **-p** option. This option has been omitted because the informational  
32254 message is not specified by this volume of IEEE Std 1003.1-2001.

## 32255 FUTURE DIRECTIONS

32256 None.

## 32257 SEE ALSO

32258 *rm*, the System Interfaces volume of IEEE Std 1003.1-2001, *remove()*, *rmdir()*, *unlink()*

## 32259 CHANGE HISTORY

32260 First released in Issue 2.

32261 **Issue 6**

32262 The normative text is reworded to avoid use of the term “must” for application requirements.

32263 **NAME**32264           sact — print current SCCS file-editing activity (**DEVELOPMENT**)32265 **SYNOPSIS**32266 xSI       sact *file...*

32267

32268 **DESCRIPTION**

32269       The *sact* utility shall inform the user of any impending deltas to a named SCCS file by writing a  
 32270       list to standard output. This situation occurs when *get -e* has been executed previously without  
 32271       a subsequent execution of *delta*, *unget*, or *sccs unedit*.

32272 **OPTIONS**

32273           None.

32274 **OPERANDS**

32275       The following operand shall be supported:

32276       *file*       A pathname of an existing SCCS file or a directory. If *file* is a directory, the *sact*  
 32277       utility shall behave as though each file in the directory were specified as a named  
 32278       file, except that non-SCCS files (last component of the pathname does not begin  
 32279       with **s.**) and unreadable files shall be silently ignored.

32280       If exactly one *file* operand appears, and it is *'-'*, the standard input shall be read;  
 32281       each line of the standard input shall be taken to be the name of an SCCS file to be  
 32282       processed. Non-SCCS files and unreadable files shall be silently ignored.

32283 **STDIN**

32284       The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each  
 32285       line of the text file shall be interpreted as an SCCS pathname.

32286 **INPUT FILES**

32287       Any SCCS files interrogated are files of an unspecified format.

32288 **ENVIRONMENT VARIABLES**32289       The following environment variables shall affect the execution of *sact*:

32290       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 32291       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 32292       Internationalization Variables for the precedence of internationalization variables  
 32293       used to determine the values of locale categories.)

32294       *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 32295       internationalization variables.

32296       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 32297       characters (for example, single-byte as opposed to multi-byte characters in  
 32298       arguments and input files).

32299       *LC\_MESSAGES*

32300       Determine the locale that should be used to affect the format and contents of  
 32301       diagnostic messages written to standard error.

32302       *NLSPATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

32303 **ASYNCHRONOUS EVENTS**

32304       Default.

**32305 STDOUT**

32306 The output for each named file shall consist of a line in the following format:

32307 "%sΔ%sΔ%sΔ%sΔ\n", <SID>, <new SID>, <login>, <date>, <time>

32308 <SID> Specifies the SID of a delta that currently exists in the SCCS file to which changes  
32309 are made to make the new delta.

32310 <new SID> Specifies the SID for the new delta to be created.

32311 <login> Contains the login name of the user who makes the delta (that is, who executed a  
32312 *get* for editing).

32313 <date> Contains the date that *get -e* was executed, in the format used by the *prs :D:* data  
32314 keyword.

32315 <time> Contains the time that *get -e* was executed, in the format used by the *prs :T:* data  
32316 keyword.

32317 If there is more than one named file or if a directory or standard input is named, each pathname  
32318 shall be written before each of the preceding lines:

32319 "\n%s:\n", <pathname>

**32320 STDERR**

32321 The standard error shall be used only for optional informative messages concerning SCCS files  
32322 with no impending deltas, and for diagnostic messages.

**32323 OUTPUT FILES**

32324 None.

**32325 EXTENDED DESCRIPTION**

32326 None.

**32327 EXIT STATUS**

32328 The following exit values shall be returned:

32329 0 Successful completion.

32330 >0 An error occurred.

**32331 CONSEQUENCES OF ERRORS**

32332 Default.

**32333 APPLICATION USAGE**

32334 None.

**32335 EXAMPLES**

32336 None.

**32337 RATIONALE**

32338 None.

**32339 FUTURE DIRECTIONS**

32340 None.

**32341 SEE ALSO**

32342 *delta*, *get*, *sccs*, *unget*

32343 **CHANGE HISTORY**

32344 First released in Issue 2.

32345 **NAME**32346        *sccs* — front end for the SCCS subsystem (**DEVELOPMENT**)32347 **SYNOPSIS**32348 xSI        *sccs* [-r] [-d *path*] [-p *path*] *command* [*options...*] [*operands...*]

32349

32350 **DESCRIPTION**32351        The *sccs* utility is a front end to the SCCS programs. It also includes the capability to run set-  
32352        user-id to another user to provide additional protection.32353        The *sccs* utility shall invoke the specified *command* with the specified *options* and *operands*. By  
32354        default, each of the *operands* shall be modified by prefixing it with the string "SCCS/s. ".32355        The *command* can be the name of one of the SCCS utilities in this volume of IEEE Std 1003.1-2001  
32356        (*admin*, *delta*, *get*, *prs*, *rmdel*, *sact*, *unget*, *val*, or *what*) or one of the pseudo-utilities listed in the  
32357        EXTENDED DESCRIPTION section.32358 **OPTIONS**32359        The *sccs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
32360        12.2, Utility Syntax Guidelines, except that *options* operands are actually options to be passed to  
32361        the utility named by *command*. When the portion of the command:32362        *command* [*options ...*] [*operands ...*]32363        is considered, all of the pseudo-utilities used as *command* shall support the Utility Syntax  
32364        Guidelines. Any of the other SCCS utilities that can be invoked in this manner support the  
32365        Guidelines to the extent indicated by their individual OPTIONS sections.32366        The following options shall be supported preceding the *command* operand:32367        **-d path**        A pathname of a directory to be used as a root directory for the SCCS files. The  
32368        default shall be the current directory. The **-d** option shall take precedence over the  
32369        *PROJECTDIR* variable. See **-p**.32370        **-p path**        A pathname of a directory in which the SCCS files are located. The default shall be  
32371        the **SCCS** directory.32372        The **-p** option differs from the **-d** option in that the **-d** option-argument shall be  
32373        prefixed to the entire pathname and the **-p** option-argument shall be inserted  
32374        before the final component of the pathname. For example:32375        *sccs -d /x -p y get a/b*

32376        converts to:

32377        *get /x/a/y/s.b*

32378        This allows the creation of aliases such as:

32379        *alias syssccs="sccs -d /usr/src"*

32380        which is used as:

32381        *syssccs get cmd/who.c*32382        **-r**            Invoke *command* with the real user ID of the process, not any effective user ID that  
32383        the *sccs* utility is set to. Certain commands (*admin*, **check**, **clean**, **diffs**, **info**, *rmdel*,  
32384        and **tell**) cannot be run set-user-ID by all users, since this would allow anyone to  
32385        change the authorizations. These commands are always run as the real user.



32386 **OPERANDS**

32387 The following operands shall be supported:

32388 *command* An SCCS utility name or the name of one of the pseudo-utilities listed in the  
 32389 EXTENDED DESCRIPTION section.

32390 *options* An option or option-argument to be passed to *command*.

32391 *operands* An operand to be passed to *command*.

32392 **STDIN**32393 See the utility description for the specified *command*.32394 **INPUT FILES**32395 See the utility description for the specified *command*.32396 **ENVIRONMENT VARIABLES**32397 The following environment variables shall affect the execution of *sccs*:

32398 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 32399 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 32400 Internationalization Variables for the precedence of internationalization variables  
 32401 used to determine the values of locale categories.)

32402 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 32403 internationalization variables.

32404 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 32405 characters (for example, single-byte as opposed to multi-byte characters in  
 32406 arguments and input files).

32407 *LC\_MESSAGES*

32408 Determine the locale that should be used to affect the format and contents of  
 32409 diagnostic messages written to standard error.

32410 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

32411 *PROJECTDIR*

32412 Provide a default value for the *-d path* option. If the value of *PROJECTDIR* begins  
 32413 with a slash, it shall be considered an absolute pathname; otherwise, the value of  
 32414 *PROJECTDIR* is treated as a user name and that user's initial working directory  
 32415 shall be examined for a subdirectory **src** or **source**. If such a directory is found, it  
 32416 shall be used. Otherwise, the value shall be used as a relative pathname.

32417 Additional environment variable effects may be found in the utility description for the specified  
 32418 *command*.

32419 **ASYNCHRONOUS EVENTS**

32420 Default.

32421 **STDOUT**32422 See the utility description for the specified *command*.32423 **STDERR**32424 See the utility description for the specified *command*.32425 **OUTPUT FILES**32426 See the utility description for the specified *command*.

## 32427 EXTENDED DESCRIPTION

32428 The following pseudo-utilities shall be supported as *command* operands. All options referred to  
 32429 in the following list are values given in the *options* operands following *command*.

32430 **check** Equivalent to **info**, except that nothing shall be printed if nothing is being edited, and a  
 32431 non-zero exit status shall be returned if anything is being edited. The intent is to have  
 32432 this included in an “install” entry in a makefile to ensure that everything is included  
 32433 into the SCCS file before a version is installed.

32434 **clean** Remove everything from the current directory that can be recreated from SCCS files,  
 32435 but do not remove any files being edited. If the **-b** option is given, branches shall be  
 32436 ignored in the determination of whether they are being edited; this is dangerous if  
 32437 branches are kept in the same directory.

32438 **create** Create an SCCS file, taking the initial contents from the file of the same name. Any  
 32439 options to *admin* are accepted. If the creation is successful, the original files shall be  
 32440 renamed by prefixing the basenames with a comma. These renamed files should be  
 32441 removed after it has been verified that the SCCS files have been created successfully.

32442 **delget** Perform a *delta* on the named files and then *get* new versions. The new versions shall  
 32443 have ID keywords expanded and shall not be editable. Any **-m**, **-p**, **-r**, **-s**, and **-y**  
 32444 options shall be passed to *delta*, and any **-b**, **-c**, **-e**, **-i**, **-k**, **-l**, **-s**, and **-x** options shall be  
 32445 passed to *get*.

32446 **deledit** Equivalent to **delget**, except that the *get* phase shall include the **-e** option. This option  
 32447 is useful for making a checkpoint of the current editing phase. The same options shall  
 32448 be passed to *delta* as described above, and all the options listed for *get* above except **-e**  
 32449 shall be passed to **edit**.

32450 **diffs** Write a difference listing between the current version of the files checked out for  
 32451 editing and the versions in SCCS format. Any **-r**, **-c**, **-i**, **-x**, and **-t** options shall be  
 32452 passed to *get*; any **-l**, **-s**, **-e**, **-f**, **-h**, and **-b** options shall be passed to *diff*. A **-C** option  
 32453 shall be passed to *diff* as **-c**.

32454 **edit** Equivalent to *get -e*.

32455 **fix** Remove the named delta, but leave a copy of the delta with the changes that were in it.  
 32456 It is useful for fixing small compiler bugs, and so on. The application shall ensure that it  
 32457 is followed by a **-r SID** option. Since **fix** does not leave audit trails, it should be used  
 32458 carefully.

32459 **info** Write a listing of all files being edited. If the **-b** option is given, branches (that is, SIDs  
 32460 with two or fewer components) shall be ignored. If a **-u user** option is given, then only  
 32461 files being edited by the named user shall be listed. A **-U** option shall be equivalent to  
 32462 **-u<current user>**.

32463 **print** Write out verbose information about the named files, equivalent to *scs prs*.

32464 **tell** Write a <newline>-separated list of the files being edited to standard output. Takes the  
 32465 **-b**, **-u**, and **-U** options like **info** and **check**.

32466 **unedit** This is the opposite of an **edit** or a *get -e*. It should be used with caution, since any  
 32467 changes made since the *get* are lost.

## 32468 EXIT STATUS

32469 The following exit values shall be returned:

32470 0 Successful completion.

32471 >0 An error occurred.

## 32472 CONSEQUENCES OF ERRORS

32473 Default.

## 32474 APPLICATION USAGE

32475 Many of the SCCS utilities take directory names as operands as well as specific filenames. The  
 32476 pseudo-utilities supported by *sccs* are not described as having this capability, but are not  
 32477 prohibited from doing so.

## 32478 EXAMPLES

32479 1. To get a file for editing, edit it and produce a new delta:

32480 `sccs get -e file.c`

32481 `ex file.c`

32482 `sccs delta file.c`

32483 2. To get a file from another directory:

32484 `sccs -p /usr/src/sccs/s. get cc.c`

32485 or:

32486 `sccs get /usr/src/sccs/s.cc.c`

32487 3. To make a delta of a large number of files in the current directory:

32488 `sccs delta *.c`

32489 4. To get a list of files being edited that are not on branches:

32490 `sccs info -b`

32491 5. To delta everything being edited by the current user:

32492 `sccs delta $(sccs tell -U)`

32493 6. In a makefile, to get source files from an SCCS file if it does not already exist:

32494 `SRCS = <list of source files>`

32495 `$(SRCS) :`

32496 `sccs get $(REL) $@`

## 32497 RATIONALE

32498 SCCS and its associated utilities are part of the XSI Development Utilities option within the XSI  
 32499 extension.

32500 SCCS is an abbreviation for Source Code Control System. It is a maintenance and enhancement  
 32501 tracking tool. When a file is put under SCCS, the source code control system maintains the file  
 32502 and, when changes are made, identifies and stores them in the file with the original source code  
 32503 and/or documentation. As other changes are made, they too are identified and retained in the  
 32504 file.

32505 Retrieval of the original and any set of changes is possible. Any version of the file as it develops  
 32506 can be reconstructed for inspection or additional modification. History data can be stored with  
 32507 each version, documenting why the changes were made, who made them, and when they were  
 32508 made.

32509 **FUTURE DIRECTIONS**

32510 None.

32511 **SEE ALSO**32512 *admin, delta, get, make, prs, rmdel, sact, unget, val, what*32513 **CHANGE HISTORY**

32514 First released in Issue 4.

32515 **Issue 6**

32516 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from  
32517 “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of  
32518 *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.

32519 The normative text is reworded to avoid use of the term “must” for application requirements.

32520 **NAME**

32521           sed — stream editor

32522 **SYNOPSIS**32523           sed [-n] *script*[*file...*]32524           sed [-n] [-e *script*]...[-f *script\_file*]...[*file...*]32525 **DESCRIPTION**

32526           The *sed* utility is a stream editor that shall read one or more text files, make editing changes  
 32527           according to a script of editing commands, and write the results to standard output. The script  
 32528           shall be obtained from either the *script* operand string or a combination of the option-arguments  
 32529           from the **-e** *script* and **-f** *script\_file* options.

32530 **OPTIONS**

32531           The *sed* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 32532           Utility Syntax Guidelines, except that the order of presentation of the **-e** and **-f** options is  
 32533           significant.

32534           The following options shall be supported:

32535           **-e** *script*     Add the editing commands specified by the *script* option-argument to the end of  
 32536                           the script of editing commands. The *script* option-argument shall have the same  
 32537                           properties as the *script* operand, described in the OPERANDS section.

32538           **-f** *script\_file* Add the editing commands in the file *script\_file* to the end of the script.

32539           **-n**           Suppress the default output (in which each line, after it is examined for editing, is  
 32540                           written to standard output). Only lines explicitly selected for output are written.

32541           Multiple **-e** and **-f** options may be specified. All commands shall be added to the script in the  
 32542           order specified, regardless of their origin.

32543 **OPERANDS**

32544           The following operands shall be supported:

32545           *file*           A pathname of a file whose contents are read and edited. If multiple *file* operands  
 32546                           are specified, the named files shall be read in the order specified and the  
 32547                           concatenation shall be edited. If no *file* operands are specified, the standard input  
 32548                           shall be used.

32549           *script*        A string to be used as the script of editing commands. The application shall not  
 32550                           present a *script* that violates the restrictions of a text file except that the final  
 32551                           character need not be a <newline>.

32552 **STDIN**

32553           The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 32554           section.

32555 **INPUT FILES**

32556           The input files shall be text files. The *script\_files* named by the **-f** option shall consist of editing  
 32557           commands.

32558 **ENVIRONMENT VARIABLES**32559           The following environment variables shall affect the execution of *sed*:

32560           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 32561                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 32562                           Internationalization Variables for the precedence of internationalization variables  
 32563                           used to determine the values of locale categories.)

32564 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 32565 internationalization variables.

32566 *LC\_COLLATE*  
 32567 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 32568 character collating elements within regular expressions.

32569 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 32570 characters (for example, single-byte as opposed to multi-byte characters in  
 32571 arguments and input files), and the behavior of character classes within regular  
 32572 expressions.

32573 *LC\_MESSAGES*  
 32574 Determine the locale that should be used to affect the format and contents of  
 32575 diagnostic messages written to standard error.

32576 xsi *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

32577 **ASYNCHRONOUS EVENTS**  
 32578 Default.

32579 **STDOUT**  
 32580 The input files shall be written to standard output, with the editing commands specified in the  
 32581 script applied. If the **-n** option is specified, only those input lines selected by the script shall be  
 32582 written to standard output.

32583 **STDERR**  
 32584 The standard error shall be used only for diagnostic messages.

32585 **OUTPUT FILES**  
 32586 The output files shall be text files whose formats are dependent on the editing commands given.

32587 **EXTENDED DESCRIPTION**  
 32588 The *script* shall consist of editing commands of the following form:  
 32589 *[address [, address] ] function*  
 32590 where *function* represents a single-character command verb from the list in **Editing Commands**  
 32591 **in sed** (on page 843), followed by any applicable arguments.  
 32592 The command can be preceded by <blank>s and/or semicolons. The function can be preceded  
 32593 by <blank>s. These optional characters shall have no effect.

32594 In default operation, *sed* cyclically shall append a line of input, less its terminating <newline>,  
 32595 into the pattern space. Normally the pattern space will be empty, unless a **D** command  
 32596 terminated the last cycle. The *sed* utility shall then apply in sequence all commands whose  
 32597 addresses select that pattern space, and at the end of the script copy the pattern space to  
 32598 standard output (except when **-n** is specified) and delete the pattern space. Whenever the  
 32599 pattern space is written to standard output or a named file, *sed* shall immediately follow it with a  
 32600 <newline>.

32601 Some of the editing commands use a hold space to save all or part of the pattern space for  
 32602 subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8 192 bytes.

**Addresses in sed**

An address is either a decimal number that counts input lines cumulatively across files, a ' '\$ ' character that addresses the last line of input, or a context address (which consists of a BRE, as described in **Regular Expressions in sed**, preceded and followed by a delimiter, usually a slash).

An editing command with no addresses shall select every pattern space.

An editing command with one address shall select each pattern space that matches the address.

An editing command with two addresses shall select the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line shall be selected.) Starting at the first line following the selected range, *sed* shall look again for the first address. Thereafter, the process shall be repeated. Omitting either or both of the address components in the following form produces undefined results:

`[address [, address] ]`

**Regular Expressions in sed**

The *sed* utility shall support the BREs described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3, Basic Regular Expressions, with the following additions:

- In a context address, the construction "`\cBREc`", where *c* is any character other than backslash or <newline>, shall be identical to "`/BRE/`". If the character designated by *c* appears following a backslash, then it shall be considered to be that literal character, which shall not terminate the BRE. For example, in the context address "`\xabc\xdefx`", the second *x* stands for itself, so that the BRE is "`abcxdef`".
- The escape sequence '`\n`' shall match a <newline> embedded in the pattern space. A literal <newline> shall not be used in the BRE of a context address or in the substitute function.
- If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in the last command applied (either as an address or as part of a substitute command) was specified.

**Editing Commands in sed**

In the following list of editing commands, the maximum number of permissible addresses for each function is indicated by `[0addr]`, `[1addr]`, or `[2addr]`, representing zero, one, or two addresses.

The argument *text* shall consist of one or more lines. Each embedded <newline> in the text shall be preceded by a backslash. Other backslashes in text shall be removed, and the following character shall be treated literally.

The **r** and **w** command verbs, and the **w** flag to the **s** command, take an optional *rfile* (or *wfile*) parameter, separated from the command verb letter or flag by one or more <blank>s; implementations may allow zero separation as an extension.

The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be created before processing begins. Implementations shall support at least ten *wfile* arguments in the script; the actual number (greater than or equal to 10) that is supported by the implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially created, if it does not exist, or shall replace the contents of an existing file.

The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following synopses indicate which arguments shall be separated from the command verbs by a single

|       |                                                                                                                                                                          |                                                                                                         |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| 32646 | <space>.                                                                                                                                                                 |                                                                                                         |
| 32647 | The <b>a</b> and <b>r</b> commands schedule text for later output. The text specified for the <b>a</b> command, and                                                      |                                                                                                         |
| 32648 | the contents of the file specified for the <b>r</b> command, shall be written to standard output just                                                                    |                                                                                                         |
| 32649 | before the next attempt to fetch a line of input when executing the <b>N</b> or <b>n</b> commands, or when                                                               |                                                                                                         |
| 32650 | reaching the end of the script. If written when reaching the end of the script, and the <b>-n</b> option                                                                 |                                                                                                         |
| 32651 | was not specified, the text shall be written after copying the pattern space to standard output.                                                                         |                                                                                                         |
| 32652 | The contents of the file specified for the <b>r</b> command shall be as of the time the output is written,                                                               |                                                                                                         |
| 32653 | not the time the <b>r</b> command is applied. The text shall be output in the order in which the <b>a</b> and <b>r</b>                                                   |                                                                                                         |
| 32654 | commands were applied to the input.                                                                                                                                      |                                                                                                         |
| 32655 | Command verbs other than { <b>a</b> , <b>b</b> , <b>c</b> , <b>i</b> , <b>r</b> , <b>t</b> , <b>w</b> , <b>:</b> , and <b>#</b> can be followed by a semicolon, optional |                                                                                                         |
| 32656 | <blank>s, and another command verb. However, when the <b>s</b> command verb is used with the <b>w</b>                                                                    |                                                                                                         |
| 32657 | flag, following it with another command in this manner produces undefined results.                                                                                       |                                                                                                         |
| 32658 | A function can be preceded by one or more '!' characters, in which case the function shall be                                                                            |                                                                                                         |
| 32659 | applied if the addresses do not select the pattern space. Zero or more <blank>s shall be accepted                                                                        |                                                                                                         |
| 32660 | before the first '!' character. It is unspecified whether <blank>s can follow a '!' character, and                                                                       |                                                                                                         |
| 32661 | conforming applications shall not follow a '!' character with <blank>s.                                                                                                  |                                                                                                         |
| 32662 | <b>[2addr]</b> {function                                                                                                                                                 |                                                                                                         |
| 32663 | function                                                                                                                                                                 |                                                                                                         |
| 32664 | ...                                                                                                                                                                      |                                                                                                         |
| 32665 | }                                                                                                                                                                        | Execute a list of <i>sed</i> functions only when the pattern space is selected. The list of             |
| 32666 |                                                                                                                                                                          | <i>sed</i> functions shall be surrounded by braces and separated by <newline>s, and                     |
| 32667 |                                                                                                                                                                          | conform to the following rules. The braces can be preceded or followed by                               |
| 32668 |                                                                                                                                                                          | <blank>s. The functions can be preceded by <blank>s, but shall not be followed                          |
| 32669 |                                                                                                                                                                          | by <blank>s. The <right-brace> shall be preceded by a <newline> and can be                              |
| 32670 |                                                                                                                                                                          | preceded or followed by <blank>s.                                                                       |
| 32671 | <b>[1addr]</b> <b>a</b> \                                                                                                                                                |                                                                                                         |
| 32672 | text                                                                                                                                                                     | Write text to standard output as described previously.                                                  |
| 32673 | <b>[2addr]</b> <b>b</b> [label]                                                                                                                                          |                                                                                                         |
| 32674 |                                                                                                                                                                          | Branch to the : function bearing the <i>label</i> . If <i>label</i> is not specified, branch to the end |
| 32675 |                                                                                                                                                                          | of the script. The implementation shall support <i>labels</i> recognized as unique up to                |
| 32676 |                                                                                                                                                                          | at least 8 characters; the actual length (greater than or equal to 8) that shall be                     |
| 32677 |                                                                                                                                                                          | supported by the implementation is unspecified. It is unspecified whether                               |
| 32678 |                                                                                                                                                                          | exceeding a label length causes an error or a silent truncation.                                        |
| 32679 | <b>[2addr]</b> <b>c</b> \                                                                                                                                                |                                                                                                         |
| 32680 | text                                                                                                                                                                     | Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range,                     |
| 32681 |                                                                                                                                                                          | place <i>text</i> on the output and start the next cycle.                                               |
| 32682 | <b>[2addr]</b> <b>d</b>                                                                                                                                                  | Delete the pattern space and start the next cycle.                                                      |
| 32683 | <b>[2addr]</b> <b>D</b>                                                                                                                                                  | Delete the initial segment of the pattern space through the first <newline> and                         |
| 32684 |                                                                                                                                                                          | start the next cycle.                                                                                   |
| 32685 | <b>[2addr]</b> <b>g</b>                                                                                                                                                  | Replace the contents of the pattern space by the contents of the hold space.                            |
| 32686 | <b>[2addr]</b> <b>G</b>                                                                                                                                                  | Append to the pattern space a <newline> followed by the contents of the hold                            |
| 32687 |                                                                                                                                                                          | space.                                                                                                  |
| 32688 | <b>[2addr]</b> <b>h</b>                                                                                                                                                  | Replace the contents of the hold space with the contents of the pattern space.                          |
| 32689 | <b>[2addr]</b> <b>H</b>                                                                                                                                                  | Append to the hold space a <newline> followed by the contents of the pattern                            |
| 32690 |                                                                                                                                                                          | space.                                                                                                  |



|       |                                       |                                                                                                    |
|-------|---------------------------------------|----------------------------------------------------------------------------------------------------|
| 32691 | <b>[1addr]i\</b>                      |                                                                                                    |
| 32692 | <i>text</i>                           | Write <i>text</i> to standard output.                                                              |
| 32693 | <b>[2addr]l</b>                       | (The letter ell.) Write the pattern space to standard output in a visually                         |
| 32694 |                                       | unambiguous form. The characters listed in the Base Definitions volume of                          |
| 32695 |                                       | IEEE Std 1003.1-2001, Table 5-1, Escape Sequences and Associated Actions ('\\',                    |
| 32696 |                                       | '\a', '\b', '\f', '\r', '\t', '\v') shall be written as the corresponding                          |
| 32697 |                                       | escape sequence; the '\n' in that table is not applicable. Non-printable characters                |
| 32698 |                                       | not in that table shall be written as one three-digit octal number (with a preceding               |
| 32699 |                                       | backslash) for each byte in the character (most significant byte first). If the size of a          |
| 32700 |                                       | byte on the system is greater than 9 bits, the format used for non-printable                       |
| 32701 |                                       | characters is implementation-defined.                                                              |
| 32702 |                                       | Long lines shall be folded, with the point of folding indicated by writing a                       |
| 32703 |                                       | backslash followed by a <newline>; the length at which folding occurs is                           |
| 32704 |                                       | unspecified, but should be appropriate for the output device. The end of each line                 |
| 32705 |                                       | shall be marked with a '\$'.                                                                       |
| 32706 | <b>[2addr]n</b>                       | Write the pattern space to standard output if the default output has not been                      |
| 32707 |                                       | suppressed, and replace the pattern space with the next line of input, less its                    |
| 32708 |                                       | terminating <newline>.                                                                             |
| 32709 |                                       | If no next line of input is available, the <b>n</b> command verb shall branch to the end of        |
| 32710 |                                       | the script and quit without starting a new cycle.                                                  |
| 32711 | <b>[2addr]N</b>                       | Append the next line of input, less its terminating <newline>, to the pattern space,               |
| 32712 |                                       | using an embedded <newline> to separate the appended material from the original                    |
| 32713 |                                       | material. Note that the current line number changes.                                               |
| 32714 |                                       | If no next line of input is available, the <b>N</b> command verb shall branch to the end of        |
| 32715 |                                       | the script and quit without starting a new cycle or copying the pattern space to                   |
| 32716 |                                       | standard output.                                                                                   |
| 32717 | <b>[2addr]p</b>                       | Write the pattern space to standard output.                                                        |
| 32718 | <b>[2addr]P</b>                       | Write the pattern space, up to the first <newline>, to standard output.                            |
| 32719 | <b>[1addr]q</b>                       | Branch to the end of the script and quit without starting a new cycle.                             |
| 32720 | <b>[1addr]r rfile</b>                 | Copy the contents of <i>rfile</i> to standard output as described previously. If <i>rfile</i> does |
| 32721 |                                       | not exist or cannot be read, it shall be treated as if it were an empty file, causing no           |
| 32722 |                                       | error condition.                                                                                   |
| 32723 | <b>[2addr]s/BRE/replacement/flags</b> |                                                                                                    |
| 32724 |                                       | Substitute the replacement string for instances of the BRE in the pattern space. Any               |
| 32725 |                                       | character other than backslash or <newline> can be used instead of a slash to                      |
| 32726 |                                       | delimit the BRE and the replacement. Within the BRE and the replacement, the                       |
| 32727 |                                       | BRE delimiter itself can be used as a literal character if it is preceded by a                     |
| 32728 |                                       | backslash.                                                                                         |
| 32729 |                                       | The replacement string shall be scanned from beginning to end. An ampersand                        |
| 32730 |                                       | (' & ') appearing in the replacement shall be replaced by the string matching the                  |
| 32731 |                                       | BRE. The special meaning of ' & ' in this context can be suppressed by preceding it                |
| 32732 |                                       | by a backslash. The characters "\n", where <i>n</i> is a digit, shall be replaced by the           |
| 32733 |                                       | text matched by the corresponding backreference expression. The special meaning                    |
| 32734 |                                       | of "\n" where <i>n</i> is a digit in this context, can be suppressed by preceding it by a          |
| 32735 |                                       | backslash. For each other backslash ('\') encountered, the following character                     |
| 32736 |                                       | shall lose its special meaning (if any). The meaning of a '\ ' immediately followed                |

|       |                                                     |                                                                                                                  |
|-------|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| 32737 |                                                     | by any character other than '&', '\', a digit, or the delimiter character used for                               |
| 32738 |                                                     | this command, is unspecified.                                                                                    |
| 32739 |                                                     | A line can be split by substituting a <newline> into it. The application shall escape                            |
| 32740 |                                                     | the <newline> in the replacement by preceding it by a backslash. A substitution                                  |
| 32741 |                                                     | shall be considered to have been performed even if the replacement string is                                     |
| 32742 |                                                     | identical to the string that it replaces. Any backslash used to alter the default                                |
| 32743 |                                                     | meaning of a subsequent character shall be discarded from the BRE or the                                         |
| 32744 |                                                     | replacement before evaluating the BRE or using the replacement.                                                  |
| 32745 |                                                     | The value of <i>flags</i> shall be zero or more of:                                                              |
| 32746 | <i>n</i>                                            | Substitute for the <i>n</i> th occurrence only of the BRE found within the                                       |
| 32747 |                                                     | pattern space.                                                                                                   |
| 32748 | <b>g</b>                                            | Globally substitute for all non-overlapping instances of the BRE                                                 |
| 32749 |                                                     | rather than just the first one. If both <b>g</b> and <i>n</i> are specified, the results                         |
| 32750 |                                                     | are unspecified.                                                                                                 |
| 32751 | <b>p</b>                                            | Write the pattern space to standard output if a replacement was                                                  |
| 32752 |                                                     | made.                                                                                                            |
| 32753 | <b>w</b> <i>wfile</i>                               | Write. Append the pattern space to <i>wfile</i> if a replacement was made.                                       |
| 32754 |                                                     | A conforming application shall precede the <i>wfile</i> argument with one                                        |
| 32755 |                                                     | or more <blank>s. If the <b>w</b> flag is not the last flag value given in a                                     |
| 32756 |                                                     | concatenation of multiple flag values, the results are undefined.                                                |
| 32757 | <b>[2addr]t</b> [ <i>label</i> ]                    |                                                                                                                  |
| 32758 |                                                     | Test. Branch to the : command verb bearing the <i>label</i> if any substitutions have been                       |
| 32759 |                                                     | made since the most recent reading of an input line or execution of a <b>t</b> . If <i>label</i> is              |
| 32760 |                                                     | not specified, branch to the end of the script.                                                                  |
| 32761 | <b>[2addr]w</b> <i>wfile</i>                        |                                                                                                                  |
| 32762 |                                                     | Append (write) the pattern space to <i>wfile</i> .                                                               |
| 32763 | <b>[2addr]x</b>                                     | Exchange the contents of the pattern and hold spaces.                                                            |
| 32764 | <b>[2addr]y</b> / <i>string1</i> / <i>string2</i> / |                                                                                                                  |
| 32765 |                                                     | Replace all occurrences of characters in <i>string1</i> with the corresponding characters                        |
| 32766 |                                                     | in <i>string2</i> . If a backslash followed by an 'n' appear in <i>string1</i> or <i>string2</i> , the two       |
| 32767 |                                                     | characters shall be handled as a single <newline>. If the number of characters in                                |
| 32768 |                                                     | <i>string1</i> and <i>string2</i> are not equal, or if any of the characters in <i>string1</i> appear more       |
| 32769 |                                                     | than once, the results are undefined. Any character other than backslash or                                      |
| 32770 |                                                     | <newline> can be used instead of slash to delimit the strings. If the delimiter is not                           |
| 32771 |                                                     | <i>n</i> , within <i>string1</i> and <i>string2</i> , the delimiter itself can be used as a literal character if |
| 32772 |                                                     | it is preceded by a backslash. If a backslash character is immediately followed by a                             |
| 32773 |                                                     | backslash character in <i>string1</i> or <i>string2</i> , the two backslash characters shall be                  |
| 32774 |                                                     | counted as a single literal backslash character. The meaning of a backslash                                      |
| 32775 |                                                     | followed by any character that is not 'n', a backslash, or the delimiter character is                            |
| 32776 |                                                     | undefined.                                                                                                       |
| 32777 | <b>[0addr]:label</b>                                | Do nothing. This command bears a <i>label</i> to which the <b>b</b> and <b>t</b> commands branch.                |
| 32778 | <b>[1addr]=</b>                                     | Write the following to standard output:                                                                          |
| 32779 |                                                     | "%d\n", <current line number>                                                                                    |
| 32780 | <b>[0addr]</b>                                      | Ignore this empty command.                                                                                       |

32781           **[0addr]#**     Ignore the '#' and the remainder of the line (treat them as a comment), with the  
 32782                           single exception that if the first two characters in the script are "#n", the default  
 32783                           output shall be suppressed; this shall be the equivalent of specifying **-n** on the  
 32784                           command line.

#### 32785 **EXIT STATUS**

32786           The following exit values shall be returned:

32787           0   Successful completion.

32788           >0   An error occurred.

#### 32789 **CONSEQUENCES OF ERRORS**

32790           Default.

#### 32791 **APPLICATION USAGE**

32792           Regular expressions match entire strings, not just individual lines, but a <newline> is matched  
 32793           by '\n' in a *sed* RE; a <newline> is not allowed by the general definition of regular expression in  
 32794           IEEE Std 1003.1-2001. Also note that '\n' cannot be used to match a <newline> at the end of an  
 32795           arbitrary input line; <newline>s appear in the pattern space as a result of the **N** editing  
 32796           command.

#### 32797 **EXAMPLES**

32798           This *sed* script simulates the BSD *cat -s* command, squeezing excess blank lines from standard  
 32799           input.

```
32800 sed -n '
32801 # Write non-empty lines.
32802 ./ {
32803 p
32804 d
32805 }
32806 # Write a single empty line, then look for more empty lines.
32807 /^$/ p
32808 # Get next line, discard the held <newline> (empty line),
32809 # and look for more empty lines.
32810 :Empty
32811 /^$/ {
32812 N
32813 s/./ /
32814 b Empty
32815 }
32816 # Write the non-empty line before going back to search
32817 # for the first in a set of empty lines.
32818 p
32819 '
```

#### 32820 **RATIONALE**

32821           This volume of IEEE Std 1003.1-2001 requires implementations to support at least ten distinct  
 32822           *wfiles*, matching historical practice on many implementations. Implementations are encouraged  
 32823           to support more, but conforming applications should not exceed this limit.

32824           The exit status codes specified here are different from those in System V. System V returns 2 for  
 32825           garbled *sed* commands, but returns zero with its usage message or if the input file could not be  
 32826           opened. The standard developers considered this to be a bug.

32827 The manner in which the **I** command writes non-printable characters was changed to avoid the  
 32828 historical backspace-overstrike method, and other requirements to achieve unambiguous output  
 32829 were added. See the RATIONALE for *ed* for details of the format chosen, which is the same as  
 32830 that chosen for *sed*.

32831 This volume of IEEE Std 1003.1-2001 requires implementations to provide pattern and hold  
 32832 spaces of at least 8192 bytes, larger than the 4000 bytes spaces used by some historical  
 32833 implementations, but less than the 20480 bytes limit used in an early proposal. Implementations  
 32834 are encouraged to allocate dynamically larger pattern and hold spaces as needed.

32835 The requirements for acceptance of <blank>s and <space>s in command lines has been made  
 32836 more explicit than in early proposals to describe clearly the historical practice and to remove  
 32837 confusion about the phrase “protect initial blanks [sic] and tabs from the stripping that is done  
 32838 on every script line” that appears in much of the historical documentation of the *sed* utility  
 32839 description of text. (Not all implementations are known to have stripped <blank>s from text  
 32840 lines, although they all have allowed leading <blank>s preceding the address on a command  
 32841 line.)

32842 The treatment of ‘#’ comments differs from the SVID which only allows a comment as the first  
 32843 line of the script, but matches BSD-derived implementations. The comment character is treated  
 32844 as a command, and it has the same properties in terms of being accepted with leading <blank>s;  
 32845 the BSD implementation has historically supported this.

32846 Early proposals required that a *script\_file* have at least one non-comment line. Some historical  
 32847 implementations have behaved in unexpected ways if this were not the case. The standard  
 32848 developers considered that this was incorrect behavior and that application developers should  
 32849 not have to avoid this feature. A correct implementation of this volume of IEEE Std 1003.1-2001  
 32850 shall permit *script\_files* that consist only of comment lines.

32851 Early proposals indicated that if **-e** and **-f** options were intermixed, all **-e** options were  
 32852 processed before any **-f** options. This has been changed to process them in the order presented  
 32853 because it matches historical practice and is more intuitive.

32854 The treatment of the **p** flag to the **s** command differs between System V and BSD-based systems  
 32855 when the default output is suppressed. In the two examples:

```
32856 echo a | sed 's/a/A/p'
```

```
32857 echo a | sed -n 's/a/A/p'
```

32858 this volume of IEEE Std 1003.1-2001, BSD, System V documentation, and the SVID indicate that  
 32859 the first example should write two lines with **A**, whereas the second should write one. Some  
 32860 System V systems write the **A** only once in both examples because the **p** flag is ignored if the **-n**  
 32861 option is not specified.

32862 This is a case of a diametrical difference between systems that could not be reconciled through  
 32863 the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V  
 32864 documentation behavior was adopted for this volume of IEEE Std 1003.1-2001 because:

- 32865 • No known documentation for any historic system describes the interaction between the **p**  
 32866 flag and the **-n** option.
- 32867 • The selected behavior is more correct as there is no technical justification for any interaction  
 32868 between the **p** flag and the **-n** option. A relationship between **-n** and the **p** flag might imply  
 32869 that they are only used together, but this ignores valid scripts that interrupt the cyclical  
 32870 nature of the processing through the use of the **D**, **d**, **q**, or branching commands. Such scripts  
 32871 rely on the **p** suffix to write the pattern space because they do not make use of the default  
 32872 output at the “bottom” of the script.

32873       • Because the **-n** option makes the **p** flag unnecessary, any interaction would only be useful if  
 32874 *sed* scripts were written to run both with and without the **-n** option. This is believed to be  
 32875 unlikely. It is even more unlikely that programmers have coded the **p** flag expecting it to be  
 32876 unnecessary. Because the interaction was not documented, the likelihood of a programmer  
 32877 discovering the interaction and depending on it is further decreased.

32878       • Finally, scripts that break under the specified behavior produce too much output instead of  
 32879 too little, which is easier to diagnose and correct.

32880       The form of the substitute command that uses the **n** suffix was limited to the first 512 matches in  
 32881 an early proposal. This limit has been removed because there is no reason an editor processing  
 32882 lines of {LINE\_MAX} length should have this restriction. The command **s/a/A/2047** should be  
 32883 able to substitute the 2047th occurrence of **a** on a line.

32884       The **b**, **t**, and **:** commands are documented to ignore leading white space, but no mention is  
 32885 made of trailing white space. Historical implementations of *sed* assigned different locations to  
 32886 the labels '**x**' and "**x**". This is not useful, and leads to subtle programming errors, but it is  
 32887 historical practice, and changing it could theoretically break working scripts. Implementors are  
 32888 encouraged to provide warning messages about labels that are never used or jumps to labels  
 32889 that do not exist.

32890       Historically, the *sed* **!** and **}** editing commands did not permit multiple commands on a single  
 32891 line using a semicolon as a command delimiter. Implementations are permitted, but not  
 32892 required, to support this extension.

#### 32893 **FUTURE DIRECTIONS**

32894       None.

#### 32895 **SEE ALSO**

32896       *awk*, *ed*, *grep*

#### 32897 **CHANGE HISTORY**

32898       First released in Issue 2.

#### 32899 **Issue 5**

32900       The FUTURE DIRECTIONS section is added.

#### 32901 **Issue 6**

32902       The following new requirements on POSIX implementations derive from alignment with the  
 32903 Single UNIX Specification:

32904       • Implementations are required to support at least ten *wfile* arguments in an editing command.

32905       The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.

32906       IEEE PASC Interpretation 1003.2 #190 is applied.

32907       IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the meaning of the backslash escape  
 32908 sequences in a replacement string for a BRE.

32909 **NAME**

32910 sh — shell, the standard command language interpreter

32911 **SYNOPSIS**32912 sh [-abCefhimnuvx] [-o *option*] [+abCefhimnuvx] [+o *option*]  
32913 [*command\_file* [*argument...*]]32914 sh -c [-abCefhimnuvx] [-o *option*] [+abCefhimnuvx] [+o *option*] *command\_string*  
32915 [*command\_name* [*argument...*]]32916 sh -s [-abCefhimnuvx] [-o *option*] [+abCefhimnuvx] [+o *option*] [*argument*]32917 **DESCRIPTION**32918 The *sh* utility is a command language interpreter that shall execute commands read from a  
32919 command line string, the standard input, or a specified file. The application shall ensure that the  
32920 commands to be executed are expressed in the language described in Chapter 2 (on page 29).

32921 Pathname expansion shall not fail due to the size of a file.

32922 Shell input and output redirections have an implementation-defined offset maximum that is  
32923 established in the open file description.32924 **OPTIONS**32925 The *sh* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
32926 Utility Syntax Guidelines, with an extension for support of a leading plus sign ('+') as noted  
32927 below.32928 The **-a**, **-b**, **-C**, **-e**, **-f**, **-m**, **-n**, **-o option**, **-u**, **-v**, and **-x** options are described as part of the *set*  
32929 utility in Section 2.14 (on page 64). The option letters derived from the *set* special built-in shall  
32930 also be accepted with a leading plus sign ('+') instead of a leading hyphen (meaning the reverse  
32931 case of the option as described in this volume of IEEE Std 1003.1-2001).

32932 The following additional options shall be supported:

32933 **-c** Read commands from the *command\_string* operand. Set the value of special  
32934 parameter 0 (see Section 2.5.2 (on page 34)) from the value of the *command\_name*  
32935 operand and the positional parameters (\$1, \$2, and so on) in sequence from the  
32936 remaining *argument* operands. No commands shall be read from the standard  
32937 input.32938 **-i** Specify that the shell is *interactive*; see below. An implementation may treat  
32939 specifying the **-i** option as an error if the real user ID of the calling process does  
32940 not equal the effective user ID or if the real group ID does not equal the effective  
32941 group ID.32942 **-s** Read commands from the standard input.32943 If there are no operands and the **-c** option is not specified, the **-s** option shall be assumed.32944 If the **-i** option is present, or if there are no operands and the shell's standard input and standard  
32945 error are attached to a terminal, the shell is considered to be *interactive*.32946 **OPERANDS**

32947 The following operands shall be supported:

32948 **-** A single hyphen shall be treated as the first operand and then ignored. If both '-'  
32949 and "--" are given as arguments, or if other operands precede the single hyphen,  
32950 the results are undefined.32951 *argument* The positional parameters (\$1, \$2, and so on) shall be set to *arguments*, if any.

32952 *command\_file* The pathname of a file containing commands. If the pathname contains one or  
 32953 more slash characters, the implementation attempts to read that file; the file need  
 32954 not be executable. If the pathname does not contain a slash character:

32955       • The implementation shall attempt to read that file from the current working  
 32956 directory; the file need not be executable.

32957       • If the file is not in the current working directory, the implementation may  
 32958 perform a search for an executable file using the value of *PATH*, as described in  
 32959 Section 2.9.1.1 (on page 48).

32960 Special parameter 0 (see Section 2.5.2 (on page 34)) shall be set to the value of  
 32961 *command\_file*. If *sh* is called using a synopsis form that omits *command\_file*, special  
 32962 parameter 0 shall be set to the value of the first argument passed to *sh* from its  
 32963 parent (for example, *argv*[0] for a C program), which is normally a pathname used  
 32964 to execute the *sh* utility.

32965 *command\_name*  
 32966 A string assigned to special parameter 0 when executing the commands in  
 32967 *command\_string*. If *command\_name* is not specified, special parameter 0 shall be set  
 32968 to the value of the first argument passed to *sh* from its parent (for example, *argv*[0]  
 32969 for a C program), which is normally a pathname used to execute the *sh* utility.

32970 *command\_string*  
 32971 A string that shall be interpreted by the shell as one or more commands, as if the  
 32972 string were the argument to the *system()* function defined in the System Interfaces  
 32973 volume of IEEE Std 1003.1-2001. If the *command\_string* operand is an empty string,  
 32974 *sh* shall exit with a zero exit status.

## 32975 **STDIN**

32976 The standard input shall be used only if one of the following is true:

- 32977       • The *-s* option is specified.
- 32978       • The *-c* option is not specified and no operands are specified.
- 32979       • The script executes one or more commands that require input from standard input (such as a  
 32980 *read* command that does not redirect its input).

32981 See the INPUT FILES section.

32982 When the shell is using standard input and it invokes a command that also uses standard input,  
 32983 the shell shall ensure that the standard input file pointer points directly after the command it has  
 32984 read when the command begins execution. It shall not read ahead in such a manner that any  
 32985 characters intended to be read by the invoked command are consumed by the shell (whether  
 32986 interpreted by the shell or not) or that characters that are not read by the invoked command are  
 32987 not seen by the shell. When the command expecting to read standard input is started  
 32988 asynchronously by an interactive shell, it is unspecified whether characters are read by the  
 32989 command or interpreted by the shell.

32990 If the standard input to *sh* is a FIFO or terminal device and is set to non-blocking reads, then *sh*  
 32991 shall enable blocking reads on standard input. This shall remain in effect when the command  
 32992 completes.

## 32993 **INPUT FILES**

32994 The input file shall be a text file, except that line lengths shall be unlimited. If the input file is  
 32995 empty or consists solely of blank lines or comments, or both, *sh* shall exit with a zero exit status.

32996 **ENVIRONMENT VARIABLES**32997 The following environment variables shall affect the execution of *sh*:

- 32998       *ENV*           This variable, when and only when an interactive shell is invoked, shall be  
32999                   subjected to parameter expansion (see Section 2.6.2 (on page 37)) by the shell, and  
33000                   the resulting value shall be used as a pathname of a file containing shell  
33001                   commands to execute in the current environment. The file need not be executable.  
33002                   If the expanded value of *ENV* is not an absolute pathname, the results are  
33003                   unspecified. *ENV* shall be ignored if the real and effective user IDs or real and  
33004                   effective group IDs of the process are different.
- 33005       *FCEDIT*       This variable, when expanded by the shell, shall determine the default value for  
33006                   the **-e** *editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be  
33007                   used as the editor. This volume of IEEE Std 1003.1-2001 specifies the effects of this  
33008                   variable only for systems supporting the User Portability Utilities option.
- 33009       *HISTFILE*      Determine a pathname naming a command history file. If the *HISTFILE* variable is  
33010                   not set, the shell may attempt to access or create a file **.sh\_history** in the directory  
33011                   referred to by the *HOME* environment variable. If the shell cannot obtain both read  
33012                   and write access to, or create, the history file, it shall use an unspecified  
33013                   mechanism that allows the history to operate properly. (References to history  
33014                   “file” in this section shall be understood to mean this unspecified mechanism in  
33015                   such cases.) An implementation may choose to access this variable only when  
33016                   initializing the history file; this initialization shall occur when *fc* or *sh* first attempt  
33017                   to retrieve entries from, or add entries to, the file, as the result of commands issued  
33018                   by the user, the file named by the *ENV* variable, or implementation-defined system  
33019                   start-up files. Implementations may choose to disable the history list mechanism  
33020                   for users with appropriate privileges who do not set *HISTFILE*; the specific  
33021                   circumstances under which this occurs are implementation-defined. If more than  
33022                   one instance of the shell is using the same history file, it is unspecified how  
33023                   updates to the history file from those shells interact. As entries are deleted from  
33024                   the history file, they shall be deleted oldest first. It is unspecified when history file  
33025                   entries are physically removed from the history file. This volume of  
33026                   IEEE Std 1003.1-2001 specifies the effects of this variable only for systems  
33027                   supporting the User Portability Utilities option.
- 33028       *HISTSIZE*      Determine a decimal number representing the limit to the number of previous  
33029                   commands that are accessible. If this variable is unset, an unspecified default  
33030                   greater than or equal to 128 shall be used. The maximum number of commands in  
33031                   the history list is unspecified, but shall be at least 128. An implementation may  
33032                   choose to access this variable only when initializing the history file, as described  
33033                   under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*  
33034                   after the history file has been initialized are effective.
- 33035       *HOME*          Determine the pathname of the user's home directory. The contents of *HOME* are  
33036                   used in tilde expansion as described in Section 2.6.1 (on page 37). This volume of  
33037                   IEEE Std 1003.1-2001 specifies the effects of this variable only for systems  
33038                   supporting the User Portability Utilities option.
- 33039       *IFS*           (Input Field Separators.) A string treated as a list of characters that shall be used  
33040                   for field splitting and to split lines into words with the *read* command. See Section  
33041                   2.6.5 (on page 42). If *IFS* is not set, the shell shall behave as if the value of *IFS* were  
33042                   <space>, <tab>, and <newline>. Implementations may ignore the value of *IFS* in  
33043                   the environment at the time *sh* is invoked, treating *IFS* as if it were not set.



|       |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 33044 | <i>LANG</i>        | Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 33045 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33046 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33047 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33048 | <i>LC_ALL</i>      | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33049 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33050 | <i>LC_COLLATE</i>  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33051 |                    | Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 33052 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33053 | <i>LC_CTYPE</i>    | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), which characters are defined as letters (character class <b>alpha</b> ), and the behavior of character classes within pattern matching.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 33054 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33055 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33056 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33057 | <i>LC_MESSAGES</i> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33058 |                    | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 33059 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33060 | <i>MAIL</i>        | Determine a pathname of the user's mailbox file for purposes of incoming mail notification. If this variable is set, the shell shall inform the user if the file named by the variable is created or if its modification time has changed. Informing the user shall be accomplished by writing a string of unspecified format to standard error prior to the writing of the next primary prompt string. Such check shall be performed only after the completion of the interval defined by the <i>MAILCHECK</i> variable after the last such check. The user shall be informed only if <i>MAIL</i> is set and <i>MAILPATH</i> is not set. This volume of IEEE Std 1003.1-2001 specifies the effects of this variable only for systems supporting the User Portability Utilities option. |
| 33061 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33062 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33063 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33064 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33065 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33066 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33067 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33068 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33069 | <i>MAILCHECK</i>   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33070 |                    | Establish a decimal integer value that specifies how often (in seconds) the shell shall check for the arrival of mail in the files specified by the <i>MAILPATH</i> or <i>MAIL</i> variables. The default value shall be 600 seconds. If set to zero, the shell shall check before issuing each primary prompt. This volume of IEEE Std 1003.1-2001 specifies the effects of this variable only for systems supporting the User Portability Utilities option.                                                                                                                                                                                                                                                                                                                           |
| 33071 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33072 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33073 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33074 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33075 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33076 | <i>MAILPATH</i>    | Provide a list of pathnames and optional messages separated by colons. If this variable is set, the shell shall inform the user if any of the files named by the variable are created or if any of their modification times change. (See the preceding entry for <i>MAIL</i> for descriptions of mail arrival and user informing.) Each pathname can be followed by ' <i>%</i> ' and a string that shall be subjected to parameter expansion and written to standard error when the modification time changes. If a ' <i>%</i> ' character in the pathname is preceded by a backslash, it shall be treated as a literal ' <i>%</i> ' in the pathname. The default message is unspecified.                                                                                               |
| 33077 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33078 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33079 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33080 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33081 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33082 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33083 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33084 |                    | The <i>MAILPATH</i> environment variable takes precedence over the <i>MAIL</i> variable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33085 |                    | This volume of IEEE Std 1003.1-2001 specifies the effects of this variable only for systems supporting the User Portability Utilities option.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 33086 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33087 | <i>NLSPATH</i>     | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 33088 | <i>PATH</i>        | Establish a string formatted as described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables, used to effect command interpretation; see Section 2.9.1.1 (on page 48).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 33089 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33090 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

33091        *PWD*        This variable shall represent an absolute pathname of the current working  
33092        directory. Assignments to this variable may be ignored unless the value is an  
33093        absolute pathname of the current working directory and there are no filename  
33094        components of dot or dot-dot.

### 33095 **ASYNCHRONOUS EVENTS**

33096        Default.

### 33097 **STDOUT**

33098        See the STDERR section.

### 33099 **STDERR**

33100        Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode),  
33101        standard error shall be used only for diagnostic messages.

### 33102 **OUTPUT FILES**

33103        None.

### 33104 **EXTENDED DESCRIPTION**

33105        See Chapter 2. The following additional capabilities are supported on systems supporting the  
33106        User Portability Utilities option.

#### 33107        **Command History List**

33108        When the *sh* utility is being used interactively, it shall maintain a list of commands previously  
33109        entered from the terminal in the file named by the *HISTFILE* environment variable. The type,  
33110        size, and internal format of this file are unspecified. Multiple *sh* processes can share access to the  
33111        file for a user, if file access permissions allow this; see the description of the *HISTFILE*  
33112        environment variable.

#### 33113        **Command Line Editing**

33114        When *sh* is being used interactively from a terminal, the current command and the command  
33115        history (see *fc*) can be edited using *vi*-mode command line editing. This mode uses commands,  
33116        described below, similar to a subset of those described in the *vi* utility. Implementations may  
33117        offer other command line editing modes corresponding to other editing utilities.

33118        The command *set -o vi* shall enable *vi*-mode editing and place *sh* into *vi* insert mode (see  
33119        **Command Line Editing (vi-mode)** (on page 855)). This command also shall disable any other  
33120        editing mode that the implementation may provide. The command *set +o vi* disables *vi*-mode  
33121        editing.

33122        Certain block-mode terminals may be unable to support shell command line editing. If a  
33123        terminal is unable to provide either edit mode, it need not be possible to *set -o vi* when using the  
33124        shell on this terminal.

33125        In the following sections, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the  
33126        *stty* utility.

## 33127 **Command Line Editing (vi-mode)**

33128 In *vi* editing mode, there shall be a distinguished line, the edit line. All the editing operations  
33129 which modify a line affect the edit line. The edit line is always the newest line in the command  
33130 history buffer.

33131 With *vi*-mode enabled, *sh* can be switched between insert mode and command mode.

33132 When in insert mode, an entered character shall be inserted into the command line, except as  
33133 noted in **vi Line Editing Insert Mode**. Upon entering *sh* and after termination of the previous  
33134 command, *sh* shall be in insert mode.

33135 Typing an escape character shall switch *sh* into command mode (see **vi Line Editing Command**  
33136 **Mode** (on page 856)). In command mode, an entered character shall either invoke a defined  
33137 operation, be used as part of a multi-character operation, or be treated as an error. A character  
33138 that is not recognized as part of an editing command shall terminate any specific editing  
33139 command and shall alert the terminal. Typing the *interrupt* character in command mode shall  
33140 cause *sh* to terminate command line editing on the current command line, reissue the prompt on  
33141 the next line of the terminal, and reset the command history (see *fc*) so that the most recently  
33142 executed command is the previous command (that is, the command that was being edited when  
33143 it was interrupted is not reentered into the history).

33144 In the following sections, the phrase “move the cursor to the beginning of the word” shall mean  
33145 “move the cursor to the first character of the current word” and the phrase “move the cursor to  
33146 the end of the word” shall mean “move the cursor to the last character of the current word”. The  
33147 phrase “beginning of the command line” indicates the point between the end of the prompt  
33148 string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and  
33149 the first character of the command text.

## 33150 **vi Line Editing Insert Mode**

33151 While in insert mode, any character typed shall be inserted in the current command line, unless  
33152 it is from the following set.

33153 <newline> Execute the current command line. If the current command line is not empty, this  
33154 line shall be entered into the command history (see *fc*).

33155 *erase* Delete the character previous to the current cursor position and move the current  
33156 cursor position back one character. In insert mode, characters shall be erased from  
33157 both the screen and the buffer when backspacing.

33158 *interrupt* Terminate command line editing with the same effects as described for  
33159 interrupting command mode; see **Command Line Editing (vi-mode)**.

33160 *kill* Clear all the characters from the input line.

33161 <control>-V Insert the next character input, even if the character is otherwise a special insert  
33162 mode character.

33163 <control>-W Delete the characters from the one preceding the cursor to the preceding word  
33164 boundary. The word boundary in this case is the closer to the cursor of either the  
33165 beginning of the line or a character that is in neither the **blank** nor **punct** character  
33166 classification of the current locale.

33167 *end-of-file* Interpreted as the end of input in *sh*. This interpretation shall occur only at the  
33168 beginning of an input line. If *end-of-file* is entered other than at the beginning of the  
33169 line, the results are unspecified.

33170 <ESC> Place *sh* into command mode.

### 33171 **vi Line Editing Command Mode**

33172 In command mode for the command line editing feature, decimal digits not beginning with 0  
33173 that precede a command letter shall be remembered. Some commands use these decimal digits  
33174 as a count number that affects the operation.

33175 The term *motion command* represents one of the commands:

33176 <space> 0 b F l W ^ \$ ; E f T w | , B e h t

33177 If the current line is not the edit line, any command that modifies the current line shall cause the  
33178 content of the current line to replace the content of the edit line, and the current line shall  
33179 become the edit line. This replacement cannot be undone (see the **u** and **U** commands below).  
33180 The modification requested shall then be performed to the edit line. When the current line is the  
33181 edit line, the modification shall be done directly to the edit line.

33182 Any command that is preceded by *count* shall take a count (the numeric value of any preceding  
33183 decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat  
33184 by the number of times specified by the count. Also unless otherwise noted, a *count* that is out of  
33185 range is considered an error condition and shall alert the terminal, but neither the cursor  
33186 position, nor the command line, shall change.

33187 The terms *word* and *bigword* are used as defined in the *vi* description. The term *save buffer*  
33188 corresponds to the term *unnamed buffer* in *vi*.

33189 The following commands shall be recognized in command mode:

33190 <newline> Execute the current command line. If the current command line is not empty, this  
33191 line shall be entered into the command history (see *fc*).

33192 <control>-L Redraw the current command line. Position the cursor at the same location on the  
33193 redrawn line.

33194 **#** Insert the character '**#**' at the beginning of the current command line and treat the  
33195 resulting edit line as a comment. This line shall be entered into the command  
33196 history; see *fc*.

33197 **=** Display the possible shell word expansions (see Section 2.6 (on page 36)) of the  
33198 bigword at the current command line position.

33199 **Note:** This does not modify the content of the current line, and therefore does not  
33200 cause the current line to become the edit line.

33201 These expansions shall be displayed on subsequent terminal lines. If the bigword  
33202 contains none of the characters '?', '\*', or '[', an asterisk ('\*') shall be  
33203 implicitly assumed at the end. If any directories are matched, these expansions  
33204 shall have a '/' character appended. After the expansion, the line shall be  
33205 redrawn, the cursor repositioned at the current cursor position, and *sh* shall be  
33206 placed in command mode.

33207 **\** Perform pathname expansion (see Section 2.6.6 (on page 42)) on the current  
33208 bigword, up to the largest set of characters that can be matched uniquely. If the  
33209 bigword contains none of the characters '?', '\*', or '[', an asterisk ('\*') shall  
33210 be implicitly assumed at the end. This maximal expansion then shall replace the  
33211 original bigword in the command line, and the cursor shall be placed after this  
33212 expansion. If the resulting bigword completely and uniquely matches a directory, a  
33213 '/' character shall be inserted directly after the bigword. If some other file is  
33214 completely matched, a single <space> shall be inserted after the bigword. After

|       |                             |                                                                                                        |
|-------|-----------------------------|--------------------------------------------------------------------------------------------------------|
| 33215 |                             | this operation, <i>sh</i> shall be placed in insert mode.                                              |
| 33216 | <b>*</b>                    | Perform pathname expansion on the current bigword and insert all expansions                            |
| 33217 |                             | into the command to replace the current bigword, with each expansion separated                         |
| 33218 |                             | by a single <space>. If at the end of the line, the current cursor position shall be                   |
| 33219 |                             | moved to the first column position following the expansions and <i>sh</i> shall be placed              |
| 33220 |                             | in insert mode. Otherwise, the current cursor position shall be the last column                        |
| 33221 |                             | position of the first character after the expansions and <i>sh</i> shall be placed in insert           |
| 33222 |                             | mode. If the current bigword contains none of the characters '?', '*', or '[',                         |
| 33223 |                             | before the operation, an asterisk shall be implicitly assumed at the end.                              |
| 33224 | <b>@letter</b>              | Insert the value of the alias named <i>_letter</i> . The symbol <i>letter</i> represents a single      |
| 33225 |                             | alphabetic character from the portable character set; implementations may support                      |
| 33226 |                             | additional characters as an extension. If the alias <i>_letter</i> contains other editing              |
| 33227 |                             | commands, these commands shall be performed as part of the insertion. If no alias                      |
| 33228 |                             | <i>_letter</i> is enabled, this command shall have no effect.                                          |
| 33229 | <b>[count]~</b>             | Convert, if the current character is a lowercase letter, to the equivalent uppercase                   |
| 33230 |                             | letter and <i>vice versa</i> , as prescribed by the current locale. The current cursor position        |
| 33231 |                             | then shall be advanced by one character. If the cursor was positioned on the last                      |
| 33232 |                             | character of the line, the case conversion shall occur, but the cursor shall not                       |
| 33233 |                             | advance. If the '~' command is preceded by a <i>count</i> , that number of characters                  |
| 33234 |                             | shall be converted, and the cursor shall be advanced to the character position after                   |
| 33235 |                             | the last character converted. If the <i>count</i> is larger than the number of characters              |
| 33236 |                             | after the cursor, this shall not be considered an error; the cursor shall advance to                   |
| 33237 |                             | the last character on the line.                                                                        |
| 33238 | <b>[count].</b>             | Repeat the most recent non-motion command, even if it was executed on an earlier                       |
| 33239 |                             | command line. If the previous command was preceded by a <i>count</i> , and no count is                 |
| 33240 |                             | given on the '.' command, the count from the previous command shall be                                 |
| 33241 |                             | included as part of the repeated command. If the '.' command is preceded by a                          |
| 33242 |                             | <i>count</i> , this shall override any <i>count</i> argument to the previous command. The <i>count</i> |
| 33243 |                             | specified in the '.' command shall become the count for subsequent '.'                                 |
| 33244 |                             | commands issued without a count.                                                                       |
| 33245 | <b>[number]v</b>            | Invoke the <i>vi</i> editor to edit the current command line in a temporary file. When the             |
| 33246 |                             | editor exits, the commands in the temporary file shall be executed and placed in                       |
| 33247 |                             | the command history. If a <i>number</i> is included, it specifies the command number in                |
| 33248 |                             | the command history to be edited, rather than the current command line.                                |
| 33249 | <b>[count]l</b> (ell)       |                                                                                                        |
| 33250 | <b>[count]&lt;space&gt;</b> |                                                                                                        |
| 33251 |                             | Move the current cursor position to the next character position. If the cursor was                     |
| 33252 |                             | positioned on the last character of the line, the terminal shall be alerted and the                    |
| 33253 |                             | cursor shall not be advanced. If the <i>count</i> is larger than the number of characters              |
| 33254 |                             | after the cursor, this shall not be considered an error; the cursor shall advance to                   |
| 33255 |                             | the last character on the line.                                                                        |
| 33256 | <b>[count]h</b>             | Move the current cursor position to the <i>count</i> th (default 1) previous character                 |
| 33257 |                             | position. If the cursor was positioned on the first character of the line, the terminal                |
| 33258 |                             | shall be alerted and the cursor shall not be moved. If the count is larger than the                    |
| 33259 |                             | number of characters before the cursor, this shall not be considered an error; the                     |
| 33260 |                             | cursor shall move to the first character on the line.                                                  |
| 33261 | <b>[count]w</b>             | Move to the start of the next word. If the cursor was positioned on the last                           |
| 33262 |                             | character of the line, the terminal shall be alerted and the cursor shall not be                       |

|       |                  |                                                                                               |
|-------|------------------|-----------------------------------------------------------------------------------------------|
| 33263 |                  | advanced. If the <i>count</i> is larger than the number of words after the cursor, this shall |
| 33264 |                  | not be considered an error; the cursor shall advance to the last character on the             |
| 33265 |                  | line.                                                                                         |
| 33266 | <b>[count]W</b>  | Move to the start of the next bigword. If the cursor was positioned on the last               |
| 33267 |                  | character of the line, the terminal shall be alerted and the cursor shall not be              |
| 33268 |                  | advanced. If the <i>count</i> is larger than the number of bigwords after the cursor, this    |
| 33269 |                  | shall not be considered an error; the cursor shall advance to the last character on           |
| 33270 |                  | the line.                                                                                     |
| 33271 | <b>[count]e</b>  | Move to the end of the current word. If at the end of a word, move to the end of the          |
| 33272 |                  | next word. If the cursor was positioned on the last character of the line, the                |
| 33273 |                  | terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger |
| 33274 |                  | than the number of words after the cursor, this shall not be considered an error; the         |
| 33275 |                  | cursor shall advance to the last character on the line.                                       |
| 33276 | <b>[count]E</b>  | Move to the end of the current bigword. If at the end of a bigword, move to the               |
| 33277 |                  | end of the next bigword. If the cursor was positioned on the last character of the            |
| 33278 |                  | line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> |
| 33279 |                  | is larger than the number of bigwords after the cursor, this shall not be considered          |
| 33280 |                  | an error; the cursor shall advance to the last character on the line.                         |
| 33281 | <b>[count]b</b>  | Move to the beginning of the current word. If at the beginning of a word, move to             |
| 33282 |                  | the beginning of the previous word. If the cursor was positioned on the first                 |
| 33283 |                  | character of the line, the terminal shall be alerted and the cursor shall not be              |
| 33284 |                  | moved. If the <i>count</i> is larger than the number of words preceding the cursor, this      |
| 33285 |                  | shall not be considered an error; the cursor shall return to the first character on the       |
| 33286 |                  | line.                                                                                         |
| 33287 | <b>[count]B</b>  | Move to the beginning of the current bigword. If at the beginning of a bigword,               |
| 33288 |                  | move to the beginning of the previous bigword. If the cursor was positioned on the            |
| 33289 |                  | first character of the line, the terminal shall be alerted and the cursor shall not be        |
| 33290 |                  | moved. If the <i>count</i> is larger than the number of bigwords preceding the cursor,        |
| 33291 |                  | this shall not be considered an error; the cursor shall return to the first character on      |
| 33292 |                  | the line.                                                                                     |
| 33293 | <b>^</b>         | Move the current cursor position to the first character on the input line that is not a       |
| 33294 |                  | <blank>.                                                                                      |
| 33295 | <b>\$</b>        | Move to the last character position on the current command line.                              |
| 33296 | <b>0</b>         | (Zero.) Move to the first character position on the current command line.                     |
| 33297 | <b>[count]  </b> | Move to the <i>count</i> th character position on the current command line. If no number      |
| 33298 |                  | is specified, move to the first position. The first character position shall be               |
| 33299 |                  | numbered 1. If the count is larger than the number of characters on the line, this            |
| 33300 |                  | shall not be considered an error; the cursor shall be placed on the last character on         |
| 33301 |                  | the line.                                                                                     |
| 33302 | <b>[count]fc</b> | Move to the first occurrence of the character 'c' that occurs after the current               |
| 33303 |                  | cursor position. If the cursor was positioned on the last character of the line, the          |
| 33304 |                  | terminal shall be alerted and the cursor shall not be advanced. If the character 'c'          |
| 33305 |                  | does not occur in the line after the current cursor position, the terminal shall be           |
| 33306 |                  | alerted and the cursor shall not be moved.                                                    |
| 33307 | <b>[count]Fc</b> | Move to the first occurrence of the character 'c' that occurs before the current              |
| 33308 |                  | cursor position. If the cursor was positioned on the first character of the line, the         |
| 33309 |                  | terminal shall be alerted and the cursor shall not be moved. If the character 'c'             |

|       |                       |                                                                                                          |
|-------|-----------------------|----------------------------------------------------------------------------------------------------------|
| 33310 |                       | does not occur in the line before the current cursor position, the terminal shall be                     |
| 33311 |                       | alerted and the cursor shall not be moved.                                                               |
| 33312 | <b>[count]tc</b>      | Move to the character before the first occurrence of the character 'c' that occurs                       |
| 33313 |                       | after the current cursor position. If the cursor was positioned on the last character                    |
| 33314 |                       | of the line, the terminal shall be alerted and the cursor shall not be advanced. If the                  |
| 33315 |                       | character 'c' does not occur in the line after the current cursor position, the                          |
| 33316 |                       | terminal shall be alerted and the cursor shall not be moved.                                             |
| 33317 | <b>[count]Tc</b>      | Move to the character after the first occurrence of the character 'c' that occurs                        |
| 33318 |                       | before the current cursor position. If the cursor was positioned on the first                            |
| 33319 |                       | character of the line, the terminal shall be alerted and the cursor shall not be                         |
| 33320 |                       | moved. If the character 'c' does not occur in the line before the current cursor                         |
| 33321 |                       | position, the terminal shall be alerted and the cursor shall not be moved.                               |
| 33322 | <b>[count];</b>       | Repeat the most recent <b>f</b> , <b>F</b> , <b>t</b> , or <b>T</b> command. Any number argument on that |
| 33323 |                       | previous command shall be ignored. Errors are those described for the repeated                           |
| 33324 |                       | command.                                                                                                 |
| 33325 | <b>[count],</b>       | Repeat the most recent <b>f</b> , <b>F</b> , <b>t</b> , or <b>T</b> command. Any number argument on that |
| 33326 |                       | previous command shall be ignored. However, reverse the direction of that                                |
| 33327 |                       | command.                                                                                                 |
| 33328 | <b>a</b>              | Enter insert mode after the current cursor position. Characters that are entered                         |
| 33329 |                       | shall be inserted before the next character.                                                             |
| 33330 | <b>A</b>              | Enter insert mode after the end of the current command line.                                             |
| 33331 | <b>i</b>              | Enter insert mode at the current cursor position. Characters that are entered shall                      |
| 33332 |                       | be inserted before the current character.                                                                |
| 33333 | <b>I</b>              | Enter insert mode at the beginning of the current command line.                                          |
| 33334 | <b>R</b>              | Enter insert mode, replacing characters from the command line beginning at the                           |
| 33335 |                       | current cursor position.                                                                                 |
| 33336 | <b>[count]cmotion</b> |                                                                                                          |
| 33337 |                       | Delete the characters between the current cursor position and the cursor position                        |
| 33338 |                       | that would result from the specified motion command. Then enter insert mode                              |
| 33339 |                       | before the first character following any deleted characters. If <i>count</i> is specified, it            |
| 33340 |                       | shall be applied to the motion command. A <i>count</i> shall be ignored for the following                |
| 33341 |                       | motion commands:                                                                                         |
| 33342 |                       | 0      ^      \$      c                                                                                  |
| 33343 |                       | If the motion command is the character 'c', the current command line shall be                            |
| 33344 |                       | cleared and insert mode shall be entered. If the motion command would move the                           |
| 33345 |                       | current cursor position toward the beginning of the command line, the character                          |
| 33346 |                       | under the current cursor position shall not be deleted. If the motion command                            |
| 33347 |                       | would move the current cursor position toward the end of the command line, the                           |
| 33348 |                       | character under the current cursor position shall be deleted. If the <i>count</i> is larger              |
| 33349 |                       | than the number of characters between the current cursor position and the end of                         |
| 33350 |                       | the command line toward which the motion command would move the cursor,                                  |
| 33351 |                       | this shall not be considered an error; all of the remaining characters in the                            |
| 33352 |                       | mentioned range shall be deleted and insert mode shall be entered. If the                                |
| 33353 |                       | motion command is invalid, the terminal shall be alerted, the cursor shall not be                        |
| 33354 |                       | moved, and no text shall be deleted.                                                                     |

|       |                       |                                                                                              |
|-------|-----------------------|----------------------------------------------------------------------------------------------|
| 33355 | <b>C</b>              | Delete from the current character to the end of the line and enter insert mode at the        |
| 33356 |                       | new end-of-line.                                                                             |
| 33357 | <b>S</b>              | Clear the entire edit line and enter insert mode.                                            |
| 33358 | <b>[count]rc</b>      | Replace the current character with the character 'c'. With a number <i>count</i> ,           |
| 33359 |                       | replace the current and the following <i>count</i> −1 characters. After this command, the    |
| 33360 |                       | current cursor position shall be on the last character that was changed. If the <i>count</i> |
| 33361 |                       | is larger than the number of characters after the cursor, this shall not be considered       |
| 33362 |                       | an error; all of the remaining characters shall be changed.                                  |
| 33363 | <b>[count]_</b>       | Append a <space> after the current character position and then append the last               |
| 33364 |                       | bigword in the previous input line after the <space>. Then enter insert mode after           |
| 33365 |                       | the last character just appended. With a number <i>count</i> , append the <i>count</i> th    |
| 33366 |                       | bigword in the previous line.                                                                |
| 33367 | <b>[count]x</b>       | Delete the character at the current cursor position and place the deleted characters         |
| 33368 |                       | in the save buffer. If the cursor was positioned on the last character of the line, the      |
| 33369 |                       | character shall be deleted and the cursor position shall be moved to the previous            |
| 33370 |                       | character (the new last character). If the <i>count</i> is larger than the number of         |
| 33371 |                       | characters after the cursor, this shall not be considered an error; all the characters       |
| 33372 |                       | from the cursor to the end of the line shall be deleted.                                     |
| 33373 | <b>[count]X</b>       | Delete the character before the current cursor position and place the deleted                |
| 33374 |                       | characters in the save buffer. The character under the current cursor position shall         |
| 33375 |                       | not change. If the cursor was positioned on the first character of the line, the             |
| 33376 |                       | terminal shall be alerted, and the <b>X</b> command shall have no effect. If the line        |
| 33377 |                       | contained a single character, the <b>X</b> command shall have no effect. If the line         |
| 33378 |                       | contained no characters, the terminal shall be alerted and the cursor shall not be           |
| 33379 |                       | moved. If the <i>count</i> is larger than the number of characters before the cursor, this   |
| 33380 |                       | shall not be considered an error; all the characters from before the cursor to the           |
| 33381 |                       | beginning of the line shall be deleted.                                                      |
| 33382 | <b>[count]dmotion</b> |                                                                                              |
| 33383 |                       | Delete the characters between the current cursor position and the character                  |
| 33384 |                       | position that would result from the motion command. A number <i>count</i> repeats the        |
| 33385 |                       | motion command <i>count</i> times. If the motion command would move toward the               |
| 33386 |                       | beginning of the command line, the character under the current cursor position               |
| 33387 |                       | shall not be deleted. If the motion command is <b>d</b> , the entire current command line    |
| 33388 |                       | shall be cleared. If the <i>count</i> is larger than the number of characters between the    |
| 33389 |                       | current cursor position and the end of the command line toward which the motion              |
| 33390 |                       | command would move the cursor, this shall not be considered an error; all of the             |
| 33391 |                       | remaining characters in the aforementioned range shall be deleted. The deleted               |
| 33392 |                       | characters shall be placed in the save buffer.                                               |
| 33393 | <b>D</b>              | Delete all characters from the current cursor position to the end of the line. The           |
| 33394 |                       | deleted characters shall be placed in the save buffer.                                       |
| 33395 | <b>[count]ymotion</b> |                                                                                              |
| 33396 |                       | Yank (that is, copy) the characters from the current cursor position to the position         |
| 33397 |                       | resulting from the motion command into the save buffer. A number <i>count</i> shall be       |
| 33398 |                       | applied to the motion command. If the motion command would move toward the                   |
| 33399 |                       | beginning of the command line, the character under the current cursor position               |
| 33400 |                       | shall not be included in the set of yanked characters. If the motion command is <b>y</b> ,   |
| 33401 |                       | the entire current command line shall be yanked into the save buffer. The current            |
| 33402 |                       | cursor position shall be unchanged. If the <i>count</i> is larger than the number of         |



|       |                                |                                                                                                             |
|-------|--------------------------------|-------------------------------------------------------------------------------------------------------------|
| 33403 |                                | characters between the current cursor position and the end of the command line                              |
| 33404 |                                | toward which the motion command would move the cursor, this shall not be                                    |
| 33405 |                                | considered an error; all of the remaining characters in the aforementioned range                            |
| 33406 |                                | shall be yanked.                                                                                            |
| 33407 | <b>Y</b>                       | Yank the characters from the current cursor position to the end of the line into the                        |
| 33408 |                                | save buffer. The current character position shall be unchanged.                                             |
| 33409 | <b>[count]p</b>                | Put a copy of the current contents of the save buffer after the current cursor                              |
| 33410 |                                | position. The current cursor position shall be advanced to the last character put                           |
| 33411 |                                | from the save buffer. A <i>count</i> shall indicate how many copies of the save buffer                      |
| 33412 |                                | shall be put.                                                                                               |
| 33413 | <b>[count]P</b>                | Put a copy of the current contents of the save buffer before the current cursor                             |
| 33414 |                                | position. The current cursor position shall be moved to the last character put from                         |
| 33415 |                                | the save buffer. A <i>count</i> shall indicate how many copies of the save buffer shall be                  |
| 33416 |                                | put.                                                                                                        |
| 33417 | <b>u</b>                       | Undo the last command that changed the edit line. This operation shall not undo                             |
| 33418 |                                | the copy of any command line to the edit line.                                                              |
| 33419 | <b>U</b>                       | Undo all changes made to the edit line. This operation shall not undo the copy of                           |
| 33420 |                                | any command line to the edit line.                                                                          |
| 33421 | <b>[count]k</b>                |                                                                                                             |
| 33422 | <b>[count]–</b>                | Set the current command line to be the <i>count</i> th previous command line in the shell                   |
| 33423 |                                | command history. If <i>count</i> is not specified, it shall default to 1. The cursor shall be               |
| 33424 |                                | positioned on the first character of the new command. If a <b>k</b> or <b>–</b> command would               |
| 33425 |                                | retreat past the maximum number of commands in effect for this shell (affected by                           |
| 33426 |                                | the <i>HISTSIZE</i> environment variable), the terminal shall be alerted, and the                           |
| 33427 |                                | command shall have no effect.                                                                               |
| 33428 | <b>[count]j</b>                |                                                                                                             |
| 33429 | <b>[count]+</b>                | Set the current command line to be the <i>count</i> th next command line in the shell                       |
| 33430 |                                | command history. If <i>count</i> is not specified, it shall default to 1. The cursor shall be               |
| 33431 |                                | positioned on the first character of the new command. If a <b>j</b> or <b>+</b> command                     |
| 33432 |                                | advances past the edit line, the current command line shall be restored to the edit                         |
| 33433 |                                | line and the terminal shall be alerted.                                                                     |
| 33434 | <b>[number]G</b>               | Set the current command line to be the oldest command line stored in the shell                              |
| 33435 |                                | command history. With a number <i>number</i> , set the current command line to be the                       |
| 33436 |                                | command line <i>number</i> in the history. If command line <i>number</i> does not exist, the                |
| 33437 |                                | terminal shall be alerted and the command line shall not be changed.                                        |
| 33438 | <b>/pattern&lt;newline&gt;</b> |                                                                                                             |
| 33439 |                                | Move backwards through the command history, searching for the specified                                     |
| 33440 |                                | pattern, beginning with the previous command line. Patterns use the pattern                                 |
| 33441 |                                | matching notation described in Section 2.13 (on page 62), except that the <b>'^'</b>                        |
| 33442 |                                | character shall have special meaning when it appears as the first character of                              |
| 33443 |                                | <i>pattern</i> . In this case, the <b>'^'</b> is discarded and the characters after the <b>'^'</b> shall be |
| 33444 |                                | matched only at the beginning of a line. Commands in the command history shall                              |
| 33445 |                                | be treated as strings, not as filenames. If the pattern is not found, the current                           |
| 33446 |                                | command line shall be unchanged and the terminal is alerted. If it is found in a                            |
| 33447 |                                | previous line, the current command line shall be set to that line and the cursor                            |
| 33448 |                                | shall be set to the first character of the new command line.                                                |

33449 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If  
 33450 there is no previous non-empty pattern, the terminal shall be alerted and the  
 33451 current command line shall remain unchanged.

33452 ?*pattern*<newline>

33453 Move forwards through the command history, searching for the specified pattern,  
 33454 beginning with the next command line. Patterns use the pattern matching notation  
 33455 described in Section 2.13 (on page 62), except that the '^' character shall have  
 33456 special meaning when it appears as the first character of *pattern*. In this case, the  
 33457 '^' is discarded and the characters after the '^' shall be matched only at the  
 33458 beginning of a line. Commands in the command history shall be treated as strings,  
 33459 not as filenames. If the pattern is not found, the current command line shall be  
 33460 unchanged and the terminal alerted. If it is found in a following line, the current  
 33461 command line shall be set to that line and the cursor shall be set to the first  
 33462 character of the new command line.

33463 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If  
 33464 there is no previous non-empty pattern, the terminal shall be alerted and the  
 33465 current command line shall remain unchanged.

33466 **n** Repeat the most recent / or ? command. If there is no previous / or ?, the terminal  
 33467 shall be alerted and the current command line shall remain unchanged.

33468 **N** Repeat the most recent / or ? command, reversing the direction of the search. If  
 33469 there is no previous / or ?, the terminal shall be alerted and the current command  
 33470 line shall remain unchanged.

#### 33471 EXIT STATUS

33472 The following exit values shall be returned:

33473 0 The script to be executed consisted solely of zero or more blank lines or comments, or  
 33474 both.

33475 1-125 A non-interactive shell detected a syntax, redirection, or variable assignment error.

33476 127 A specified *command\_file* could not be found by a non-interactive shell.

33477 Otherwise, the shell shall return the exit status of the last command it invoked or attempted to  
 33478 invoke (see also the *exit* utility in Section 2.14 (on page 64)).

#### 33479 CONSEQUENCES OF ERRORS

33480 See Section 2.8.1 (on page 46).

#### 33481 APPLICATION USAGE

33482 Standard input and standard error are the files that determine whether a shell is interactive  
 33483 when -i is not specified. For example:

33484 sh > file

33485 and:

33486 sh 2> file

33487 create interactive and non-interactive shells, respectively. Although both accept terminal input,  
 33488 the results of error conditions are different, as described in Section 2.8.1 (on page 46); in the  
 33489 second example a redirection error encountered by a special built-in utility aborts the shell.

33490 A conforming application must protect its first operand, if it starts with a plus sign, by preceding  
 33491 it with the "--" argument that denotes the end of the options.

33492 Applications should note that the standard *PATH* to the shell cannot be assumed to be either  
 33493 **/bin/sh** or **/usr/bin/sh**, and should be determined by interrogation of the *PATH* returned by  
 33494 *getconf PATH*, ensuring that the returned pathname is an absolute pathname and not a shell  
 33495 built-in.

33496 For example, to determine the location of the standard *sh* utility:

33497 `command -v sh`

33498 On some implementations this might return:

33499 `/usr/xpg4/bin/sh`

33500 Furthermore, on systems that support executable scripts (the "#!" construct), it is  
 33501 recommended that applications using executable scripts install them using *getconf -v* to  
 33502 determine the shell pathname and update the "#!" script appropriately as it is being installed  
 33503 (for example, with *sed*). For example:

```

33504 #
33505 # Installation time script to install correct POSIX shell pathname
33506 #
33507 # Get list of paths to check
33508 #
33509 Sifs=$IFS
33510 IFS=:
33511 set $(getconf PATH)
33512 IFS=$Sifs
33513 #
33514 # Check each path for 'sh'
33515 #
33516 for i in $@
33517 do
33518 if [-f ${i}/sh];
33519 then
33520 Pshell=${i}/sh
33521 fi
33522 done
33523 #
33524 # This is the list of scripts to update. They should be of the
33525 # form '${name}.source' and will be transformed to '${name}'.
33526 # Each script should begin:
33527 #
33528 # !INSTALLSHELLPATH -p
33529 #
33530 scripts="a b c"
33531 #
33532 # Transform each script
33533 #
33534 for i in ${scripts}
33535 do
33536 sed -e "s|INSTALLSHELLPATH|${Pshell}|" < ${i}.source > ${i}
33537 done

```

33538 **EXAMPLES**

- 33539           1. Execute a shell command from a string:
- 33540            sh -c "cat myfile"
- 33541           2. Execute a shell script from a file in the current directory:
- 33542            sh my\_shell\_cmds

33543 **RATIONALE**

33544           The *sh* utility and the *set* special built-in utility share a common set of options.

33545           The KornShell ignores the contents of *IFS* upon entry to the script. A conforming application cannot rely on importing *IFS*. One justification for this, beyond security considerations, is to assist possible future shell compilers. Allowing *IFS* to be imported from the environment prevents many optimizations that might otherwise be performed via dataflow analysis of the script itself.

33550           The text in the STDIN section about non-blocking reads concerns an instance of *sh* that has been invoked, probably by a C-language program, with standard input that has been opened using the *O\_NONBLOCK* flag; see *open()* in the System Interfaces volume of IEEE Std 1003.1-2001. If the shell did not reset this flag, it would immediately terminate because no input data would be available yet and that would be considered the same as end-of-file.

33555           The options associated with a *restricted shell* (command name *rsh* and the *-r* option) were excluded because the standard developers considered that the implied level of security could not be achieved and they did not want to raise false expectations.

33558           On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the name *-i*. When it is called by a sequence such as:

33560           sh -

33561           or by:

33562           #! usr/bin/sh -

33563           the historical systems have assumed that no option letters follow. Thus, this volume of IEEE Std 1003.1-2001 allows the single hyphen to mark the end of the options, in addition to the use of the regular *"--"* argument, because it was considered that the older practice was so pervasive. An alternative approach is taken by the KornShell, where real and effective user/group IDs must match for an interactive shell; this behavior is specifically allowed by this volume of IEEE Std 1003.1-2001.

33569           **Note:**     There are other problems with set-user-ID scripts that the two approaches described here do not resolve.

33571           The initialization process for the history file can be dependent on the system start-up files, in that they may contain commands that effectively preempt the user's settings of *HISTFILE* and *HISTSIZE*. For example, function definition commands are recorded in the history file, unless the *set -o nolog* option is set. If the system administrator includes function definitions in some system start-up file called before the *ENV* file, the history file is initialized before the user gets a chance to influence its characteristics. In some historical shells, the history file is initialized just after the *ENV* file has been processed. Therefore, it is implementation-defined whether changes made to *HISTFILE* after the history file has been initialized are effective.

33579           The default messages for the various *MAIL*-related messages are unspecified because they vary across implementations. Typical messages are:

33580

33581 "you have mail\n"

33582 or:

33583 "you have new mail\n"

33584 It is important that the descriptions of command line editing refer to the same shell as that in  
33585 IEEE Std 1003.1-2001 so that interactive users can also be application programmers without  
33586 having to deal with programmatic differences in their two environments. It is also essential that  
33587 the utility name *sh* be specified because this explicit utility name is too firmly rooted in historical  
33588 practice of application programs for it to change.

33589 Consideration was given to mandating a diagnostic message when attempting to set *vi*-mode on  
33590 terminals that do not support command line editing. However, it is not historical practice for the  
33591 shell to be cognizant of all terminal types and thus be able to detect inappropriate terminals in  
33592 all cases. Implementations are encouraged to supply diagnostics in this case whenever possible,  
33593 rather than leaving the user in a state where editing commands work incorrectly.

33594 In early proposals, the KornShell-derived *emacs* mode of command line editing was included,  
33595 even though the *emacs* editor itself was not. The community of *emacs* proponents was adamant  
33596 that the full *emacs* editor not be standardized because they were concerned that an attempt to  
33597 standardize this very powerful environment would encourage vendors to ship strictly  
33598 conforming versions lacking the extensibility required by the community. The author of the  
33599 original *emacs* program also expressed his desire to omit the program. Furthermore, there were a  
33600 number of historical systems that did not include *emacs*, or included it without supporting it, but  
33601 there were very few that did not include and support *vi*. The shell *emacs* command line editing  
33602 mode was finally omitted because it became apparent that the KornShell version and the editor  
33603 being distributed with the GNU system had diverged in some respects. The author of *emacs*  
33604 requested that the POSIX *emacs* mode either be deleted or have a significant number of  
33605 unspecified conditions. Although the KornShell author agreed to consider changes to bring the  
33606 shell into alignment, the standard developers decided to defer specification at that time. At the  
33607 time, it was assumed that convergence on an acceptable definition would occur for a subsequent  
33608 draft, but that has not happened, and there appears to be no impetus to do so. In any case,  
33609 implementations are free to offer additional command line editing modes based on the exact  
33610 models of editors their users are most comfortable with.

33611 Early proposals had the following list entry in **vi Line Editing Insert Mode** (on page 855):

33612 \ If followed by the *erase* or *kill* character, that character shall be inserted into the input line.  
33613 Otherwise, the backslash itself shall be inserted into the input line.

33614 However, this is not actually a feature of *sh* command line editing insert mode, but one of some  
33615 historical terminal line drivers. Some conforming implementations continue to do this when the  
33616 *stty ixtextn* flag is set.

#### 33617 FUTURE DIRECTIONS

33618 None.

#### 33619 SEE ALSO

33620 Chapter 2 (on page 29), *cd*, *echo*, *exit*, *fc*, *pwd*, *read*, *set*, *stty*, *test*, *umask*, *vi*, the System Interfaces  
33621 volume of IEEE Std 1003.1-2001, *dup()*, *exec*, *exit()*, *fork()*, *open()*, *pipe()*, *signal()*, *system()*,  
33622 *ulimit()*, *umask()*, *wait()*

#### 33623 CHANGE HISTORY

33624 First released in Issue 2.

33625 **Issue 5**

33626 The FUTURE DIRECTIONS section is added.

33627 Text is added to the DESCRIPTION for the Large File Summit proposal.

33628 **Issue 6**

33629 The Open Group Corrigendum U029/2 is applied, correcting the second SYNOPSIS.

33630 The Open Group Corrigendum U027/3 is applied, correcting a typographical error.

33631 The following new requirements on POSIX implementations derive from alignment with the  
33632 Single UNIX Specification:

33633 • The option letters derived from the *set* special built-in are also accepted with a leading plus  
33634 sign ('+').

33635 • Large file extensions are added:

33636 — Pathname expansion does not fail due to the size of a file.

33637 — Shell input and output redirections have an implementation-defined offset maximum  
33638 that is established in the open file description.

33639 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to  
33640 “directory referred to by the *HOME* environment variable”.

33641 Descriptions for the *ENV* and *PWD* environment variables are included to align with the  
33642 IEEE P1003.2b draft standard.

33643 The normative text is reworded to avoid use of the term “must” for application requirements.

33644 **NAME**

33645           sleep — suspend execution for an interval

33646 **SYNOPSIS**33647           sleep *time*33648 **DESCRIPTION**33649           The *sleep* utility shall suspend execution for at least the integral number of seconds specified by  
33650           the *time* operand.33651 **OPTIONS**

33652           None.

33653 **OPERANDS**

33654           The following operand shall be supported:

33655           *time*           A non-negative decimal integer specifying the number of seconds for which to  
33656                               suspend execution.33657 **STDIN**

33658           Not used.

33659 **INPUT FILES**

33660           None.

33661 **ENVIRONMENT VARIABLES**33662           The following environment variables shall affect the execution of *sleep*:33663           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
33664                               (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
33665                               Internationalization Variables for the precedence of internationalization variables  
33666                               used to determine the values of locale categories.)33667           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
33668                               internationalization variables.33669           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
33670                               characters (for example, single-byte as opposed to multi-byte characters in  
33671                               arguments).33672           *LC\_MESSAGES*33673                               Determine the locale that should be used to affect the format and contents of  
33674                               diagnostic messages written to standard error.33675 *XS1*           *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.33676 **ASYNCHRONOUS EVENTS**33677           If the *sleep* utility receives a SIGALRM signal, one of the following actions shall be taken:

- 33678           1. Terminate normally with a zero exit status.
- 
- 33679           2. Effectively ignore the signal.
- 
- 33680           3. Provide the default behavior for signals described in the ASYNCHRONOUS EVENTS
- 
- 33681               section of Section 1.11 (on page 20). This could include terminating with a non-zero exit
- 
- 33682               status.

33683           The *sleep* utility shall take the standard action for all other signals.

33684 **STDOUT**

33685 Not used.

33686 **STDERR**

33687 The standard error shall be used only for diagnostic messages.

33688 **OUTPUT FILES**

33689 None.

33690 **EXTENDED DESCRIPTION**

33691 None.

33692 **EXIT STATUS**

33693 The following exit values shall be returned:

33694 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal  
33695 was received. See the ASYNCHRONOUS EVENTS section.

33696 &gt;0 An error occurred.

33697 **CONSEQUENCES OF ERRORS**

33698 Default.

33699 **APPLICATION USAGE**

33700 None.

33701 **EXAMPLES**33702 The *sleep* utility can be used to execute a command after a certain amount of time, as in:33703 (sleep 105; *command*) &

33704 or to execute a command every so often, as in:

33705 while true

33706 do

33707 *command*

33708 sleep 37

33709 done

33710 **RATIONALE**

33711 The exit status is allowed to be zero when *sleep* is interrupted by the SIGALRM signal because  
33712 most implementations of this utility rely on the arrival of that signal to notify them that the  
33713 requested finishing time has been successfully attained. Such implementations thus do not  
33714 distinguish this situation from the successful completion case. Other implementations are  
33715 allowed to catch the signal and go back to sleep until the requested time expires or to provide  
33716 the normal signal termination procedures.

33717 As with all other utilities that take integral operands and do not specify subranges of allowed  
33718 values, *sleep* is required by this volume of IEEE Std 1003.1-2001 to deal with *time* requests of up  
33719 to 2 147 483 647 seconds. This may mean that some implementations have to make multiple calls  
33720 to the delay mechanism of the underlying operating system if its argument range is less than  
33721 this.

33722 **FUTURE DIRECTIONS**

33723 None.

33724 **SEE ALSO**33725 *wait*, the System Interfaces volume of IEEE Std 1003.1-2001, *alarm()*, *sleep()*



33726 **CHANGE HISTORY**

33727 First released in Issue 2.

33728 **NAME**

33729           sort — sort, merge, or sequence check text files

33730 **SYNOPSIS**33731           sort [-m] [-o *output*] [-bdfinru] [-t *char*] [-k *keydef*]... [*file*...]33732           sort -c [-bdfinru] [-t *char*] [-k *keydef*] [*file*]33733 **DESCRIPTION**33734           The *sort* utility shall perform one of the following functions:

- 33735           1. Sort lines of all the named files together and write the result to the specified output.
- 33736           2. Merge lines of all the named (presorted) files together and write the result to the specified  
33737           output.
- 33738           3. Check that a single input file is correctly presorted.

33739           Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no  
33740           sort keys are specified, the entire line up to, but not including, the terminating <newline>), and  
33741           shall be performed using the collating sequence of the current locale.

33742 **OPTIONS**

33743           The *sort* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
33744           12.2, Utility Syntax Guidelines, and the **-k** *keydef* option should follow the **-b**, **-d**, **-f**, **-i**, **-n**, and  
33745           **-r** options.

33746           The following options shall be supported:

- 33747           **-c**           Check that the single input file is ordered as specified by the arguments and the  
33748           collating sequence of the current locale. No output shall be produced; only the exit  
33749           code shall be affected.
- 33750           **-m**           Merge only; the input file shall be assumed to be already sorted.
- 33751           **-o** *output*   Specify the name of an output file to be used instead of the standard output. This  
33752           file can be the same as one of the input *files*.
- 33753           **-u**           Unique: suppress all but one in each set of lines having equal keys. If used with  
33754           the **-c** option, check that there are no lines with duplicate keys, in addition to  
33755           checking that the input file is sorted.

33756           The following options shall override the default ordering rules. When ordering options appear  
33757           independent of any key field specifications, the requested field ordering rules shall be applied  
33758           globally to all sort keys. When attached to a specific key (see **-k**), the specified ordering options  
33759           shall override all global ordering options for that key.

- 33760           **-d**           Specify that only <blank>s and alphanumeric characters, according to the current  
33761           setting of *LC\_CTYPE*, shall be significant in comparisons. The behavior is  
33762           undefined for a sort key to which **-i** or **-n** also applies.
- 33763           **-f**           Consider all lowercase characters that have uppercase equivalents, according to  
33764           the current setting of *LC\_CTYPE*, to be the uppercase equivalent for the purposes  
33765           of comparison.
- 33766           **-i**           Ignore all characters that are non-printable, according to the current setting of  
33767           *LC\_CTYPE*.
- 33768           **-n**           Restrict the sort key to an initial numeric string, consisting of optional <blank>s,  
33769           optional minus sign, and zero or more digits with an optional radix character and  
33770           thousands separators (as defined in the current locale), which shall be sorted by

33771 arithmetic value. An empty digit string shall be treated as zero. Leading zeros and  
 33772 signs on zeros shall not affect ordering.

33773 **-r** Reverse the sense of comparisons.

33774 The treatment of field separators can be altered using the options:

33775 **-b** Ignore leading <blank>s when determining the starting and ending positions of a  
 33776 restricted sort key. If the **-b** option is specified before the first **-k** option, it shall be  
 33777 applied to all **-k** options. Otherwise, the **-b** option can be attached independently  
 33778 to each **-k** *field\_start* or *field\_end* option-argument (see below).

33779 **-t char** Use *char* as the field separator character; *char* shall not be considered to be part of a  
 33780 field (although it can be included in a sort key). Each occurrence of *char* shall be  
 33781 significant (for example, <*char*><*char*> delimits an empty field). If **-t** is not  
 33782 specified, <blank>s shall be used as default field separators; each maximal non-  
 33783 empty sequence of <blank>s that follows a non-<blank> shall be a field separator.

33784 Sort keys can be specified using the options:

33785 **-k keydef** The *keydef* argument is a restricted sort key field definition. The format of this  
 33786 definition is:

33787 *field\_start*[*type*] [, *field\_end*[*type*]]

33788 where *field\_start* and *field\_end* define a key field restricted to a portion of the line  
 33789 (see the EXTENDED DESCRIPTION section), and *type* is a modifier from the list of  
 33790 characters 'b', 'd', 'f', 'i', 'n', 'r'. The 'b' modifier shall behave like the  
 33791 **-b** option, but shall apply only to the *field\_start* or *field\_end* to which it is attached.  
 33792 The other modifiers shall behave like the corresponding options, but shall apply  
 33793 only to the key field to which they are attached; they shall have this effect if  
 33794 specified with *field\_start*, *field\_end*, or both. If any modifier is attached to a  
 33795 *field\_start* or to a *field\_end*, no option shall apply to either. Implementations shall  
 33796 support at least nine occurrences of the **-k** option, which shall be significant in  
 33797 command line order. If no **-k** option is specified, a default sort key of the entire  
 33798 line shall be used.

33799 When there are multiple key fields, later keys shall be compared only after all  
 33800 earlier keys compare equal. Except when the **-u** option is specified, lines that  
 33801 otherwise compare equal shall be ordered as if none of the options **-d**, **-f**, **-i**, **-n**, or  
 33802 **-k** were present (but with **-r** still in effect, if it was specified) and with all bytes in  
 33803 the lines significant to the comparison. The order in which lines that still compare  
 33804 equal are written is unspecified.

## 33805 OPERANDS

33806 The following operand shall be supported:

33807 *file* A pathname of a file to be sorted, merged, or checked. If no *file* operands are  
 33808 specified, or if a *file* operand is '-', the standard input shall be used.

## 33809 STDIN

33810 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
 33811 See the INPUT FILES section.

## 33812 INPUT FILES

33813 The input files shall be text files, except that the *sort* utility shall add a <newline> to the end of a  
 33814 file ending with an incomplete last line.

33815 **ENVIRONMENT VARIABLES**

33816 The following environment variables shall affect the execution of *sort*:

33817 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 33818 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 33819 Internationalization Variables for the precedence of internationalization variables  
 33820 used to determine the values of locale categories.)

33821 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 33822 internationalization variables.

33823 *LC\_COLLATE*  
 33824 Determine the locale for ordering rules.

33825 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 33826 characters (for example, single-byte as opposed to multi-byte characters in  
 33827 arguments and input files) and the behavior of character classification for the *-b*,  
 33828 *-d*, *-f*, *-i*, and *-n* options.

33829 *LC\_MESSAGES*  
 33830 Determine the locale that should be used to affect the format and contents of  
 33831 diagnostic messages written to standard error.

33832 *LC\_NUMERIC*  
 33833 Determine the locale for the definition of the radix character and thousands  
 33834 separator for the *-n* option.

33835 *XS* *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

33836 **ASYNCHRONOUS EVENTS**

33837 Default.

33838 **STDOUT**

33839 Unless the *-o* or *-c* options are in effect, the standard output shall contain the sorted input.

33840 **STDERR**

33841 The standard error shall be used for diagnostic messages. A warning message about correcting  
 33842 an incomplete last line of an input file may be generated, but need not affect the final exit status.

33843 **OUTPUT FILES**

33844 If the *-o* option is in effect, the sorted input shall be written to the file *output*.

33845 **EXTENDED DESCRIPTION**

33846 The notation:

33847 *-k field\_start[type] [,field\_end[type]]*

33848 shall define a key field that begins at *field\_start* and ends at *field\_end* inclusive, unless *field\_start*  
 33849 falls beyond the end of the line or after *field\_end*, in which case the key field is empty. A missing  
 33850 *field\_end* shall mean the last character of the line.

33851 A field comprises a maximal sequence of non-separating characters and, in the absence of option  
 33852 *-t*, any preceding field separator.

33853 The *field\_start* portion of the *keydef* option-argument shall have the form:

33854 *field\_number[.first\_character]*

33855 Fields and characters within fields shall be numbered starting with 1. The *field\_number* and  
 33856 *first\_character* pieces, interpreted as positive decimal integers, shall specify the first character to  
 33857 be used as part of a sort key. If *.first\_character* is omitted, it shall refer to the first character of the

33858 field.

33859 The *field\_end* portion of the *keydef* option-argument shall have the form:

33860 *field\_number*[.*last\_character*]

33861 The *field\_number* shall be as described above for *field\_start*. The *last\_character* piece, interpreted  
 33862 as a non-negative decimal integer, shall specify the last character to be used as part of the sort  
 33863 key. If *last\_character* evaluates to zero or *.last\_character* is omitted, it shall refer to the last  
 33864 character of the field specified by *field\_number*.

33865 If the **-b** option or **b** type modifier is in effect, characters within a field shall be counted from the  
 33866 first non-*<blank>* in the field. (This shall apply separately to *first\_character* and *last\_character*.)

### 33867 EXIT STATUS

33868 The following exit values shall be returned:

33869 0 All input files were output successfully, or **-c** was specified and the input file was correctly  
 33870 sorted.

33871 1 Under the **-c** option, the file was not ordered as specified, or if the **-c** and **-u** options were  
 33872 both specified, two input lines were found with equal keys.

33873 >1 An error occurred.

### 33874 CONSEQUENCES OF ERRORS

33875 Default.

### 33876 APPLICATION USAGE

33877 The default value for **-t**, *<blank>*, has different properties from, for example, **-t**"*<space>*". If a  
 33878 line contains:

33879 *<space><space>foo*

33880 the following treatment would occur with default separation as opposed to specifically selecting  
 33881 a *<space>*:

| Field | Default                              | <b>-t</b> " <i>&lt;space&gt;</i> " |
|-------|--------------------------------------|------------------------------------|
| 1     | <i>&lt;space&gt;&lt;space&gt;foo</i> | <i>empty</i>                       |
| 2     | <i>empty</i>                         | <i>empty</i>                       |
| 3     | <i>empty</i>                         | <i>foo</i>                         |

33886 The leading field separator itself is included in a field when **-t** is not used. For example, this  
 33887 command returns an exit status of zero, meaning the input was already sorted:

33888 `sort -c -k 2 <<eof`  
 33889 `y<tab>b`  
 33890 `x<space>a`  
 33891 `eof`

33892 (assuming that a *<tab>* precedes the *<space>* in the current collating sequence). The field  
 33893 separator is not included in a field when it is explicitly set via **-t**. This is historical practice and  
 33894 allows usage such as:

33895 `sort -t "|" -k 2n <<eof`  
 33896 `Atlanta|425022|Georgia`  
 33897 `Birmingham|284413|Alabama`  
 33898 `Columbia|100385|South Carolina`  
 33899 `eof`

33900 where the second field can be correctly sorted numerically without regard to the non-numeric  
33901 field separator.

33902 The wording in the OPTIONS section clarifies that the **-b**, **-d**, **-f**, **-i**, **-n**, and **-r** options have to  
33903 come before the first sort key specified if they are intended to apply to all specified keys. The  
33904 way it is described in this volume of IEEE Std 1003.1-2001 matches historical practice, not  
33905 historical documentation. The results are unspecified if these options are specified after a **-k**  
33906 option.

33907 The **-f** option might not work as expected in locales where there is not a one-to-one mapping  
33908 between an uppercase and a lowercase letter.

#### 33909 EXAMPLES

33910 1. The following command sorts the contents of **infile** with the second field as the sort key:

```
33911 sort -k 2,2 infile
```

33912 2. The following command sorts, in reverse order, the contents of **infile1** and **infile2**, placing  
33913 the output in **outfile** and using the second character of the second field as the sort key  
33914 (assuming that the first character of the second field is the field separator):

```
33915 sort -r -o outfile -k 2.2,2.2 infile1 infile2
```

33916 3. The following command sorts the contents of **infile1** and **infile2** using the second non-  
33917 <blank> of the second field as the sort key:

```
33918 sort -k 2.2b,2.2b infile1 infile2
```

33919 4. The following command prints the System V password file (user database) sorted by the  
33920 numeric user ID (the third colon-separated field):

```
33921 sort -t : -k 3,3n /etc/passwd
```

33922 5. The following command prints the lines of the already sorted file **infile**, suppressing all  
33923 but one occurrence of lines having the same third field:

```
33924 sort -um -k 3.1,3.0 infile
```

#### 33925 RATIONALE

33926 Examples in some historical documentation state that options **-um** with one input file keep the  
33927 first in each set of lines with equal keys. This behavior was deemed to be an implementation  
33928 artifact and was not standardized.

33929 The **-z** option was omitted; it is not standard practice on most systems and is inconsistent with  
33930 using *sort* to sort several files individually and then merge them together. The text concerning **-z**  
33931 in historical documentation appeared to require implementations to determine the proper buffer  
33932 length during the sort phase of operation, but not during the merge.

33933 The **-y** option was omitted because of non-portability. The **-M** option, present in System V, was  
33934 omitted because of non-portability in international usage.

33935 An undocumented **-T** option exists in some implementations. It is used to specify a directory for  
33936 intermediate files. Implementations are encouraged to support the use of the *TMPDIR*  
33937 environment variable instead of adding an option to support this functionality.

33938 The **-k** option was added to satisfy two objections. First, the zero-based counting used by *sort* is  
33939 not consistent with other utility conventions. Second, it did not meet syntax guideline  
33940 requirements.

33941 Historical documentation indicates that “setting **-n** implies **-b**”. The description of **-n** already  
33942 states that optional leading <blank>s are tolerated in doing the comparison. If **-b** is enabled,

33943 rather than implied, by **-n**, this has unusual side effects. When a character offset is used in a  
33944 column of numbers (for example, to sort modulo 100), that offset is measured relative to the  
33945 most significant digit, not to the column. Based upon a recommendation from the author of the  
33946 original *sort* utility, the **-b** implication has been omitted from this volume of  
33947 IEEE Std 1003.1-2001, and an application wishing to achieve the previously mentioned side  
33948 effects has to code the **-b** flag explicitly.

33949 **FUTURE DIRECTIONS**

33950 None.

33951 **SEE ALSO**

33952 *comm, join, uniq*, the System Interfaces volume of IEEE Std 1003.1-2001, *toupper()*

33953 **CHANGE HISTORY**

33954 First released in Issue 2.

33955 **Issue 6**

33956 IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.

33957 IEEE PASC Interpretation 1003.2 #168 is applied.

33958 **NAME**

33959           split — split files into pieces

33960 **SYNOPSIS**33961 UP       split [-l *line\_count*] [-a *suffix\_length*] [*file* [*name*]]33962           split -b *n*[*k*|*m*] [-a *suffix\_length*] [*file* [*name*]]

33963

33964 **DESCRIPTION**

33965       The *split* utility shall read an input file and write one or more output files. The default size of  
 33966       each output file shall be 1 000 lines. The size of the output files can be modified by specification  
 33967       of the **-b** or **-l** options. Each output file shall be created with a unique suffix. The suffix shall  
 33968       consist of exactly *suffix\_length* lowercase letters from the POSIX locale. The letters of the suffix  
 33969       shall be used as if they were a base-26 digit system, with the first suffix to be created consisting  
 33970       of all 'a' characters, the second with a 'b' replacing the last 'a', and so on, until a name of all  
 33971       'z' characters is created. By default, the names of the output files shall be 'x', followed by a  
 33972       two-character suffix from the character set as described above, starting with "aa", "ab", "ac",  
 33973       and so on, and continuing until the suffix "zz", for a maximum of 676 files.

33974       If the number of files required exceeds the maximum allowed by the suffix length provided,  
 33975       such that the last allowable file would be larger than the requested size, the *split* utility shall fail  
 33976       after creating the last file with a valid suffix; *split* shall not delete the files it created with valid  
 33977       suffixes. If the file limit is not exceeded, the last file created shall contain the remainder of the  
 33978       input file, and may be smaller than the requested size.

33979 **OPTIONS**

33980       The *split* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 33981       12.2, Utility Syntax Guidelines.

33982       The following options shall be supported:

33983       **-a *suffix\_length***

33984           Use *suffix\_length* letters to form the suffix portion of the filenames of the split file.  
 33985           If **-a** is not specified, the default suffix length shall be two. If the sum of the *name*  
 33986           operand and the *suffix\_length* option-argument would create a filename exceeding  
 33987           {NAME\_MAX} bytes, an error shall result; *split* shall exit with a diagnostic  
 33988           message and no files shall be created.

33989       **-b *n***           Split a file into pieces *n* bytes in size.

33990       **-b *nk***           Split a file into pieces *n*\*1 024 bytes in size.

33991       **-b *nm***           Split a file into pieces *n*\*1 048 576 bytes in size.

33992       **-l *line\_count***   Specify the number of lines in each resulting file piece. The *line\_count* argument is  
 33993       an unsigned decimal integer. The default is 1 000. If the input does not end with a  
 33994       <newline>, the partial line shall be included in the last output file.

33995 **OPERANDS**

33996       The following operands shall be supported:

33997       *file*           The pathname of the ordinary file to be split. If no input file is given or *file* is '-',  
 33998       the standard input shall be used.

33999       *name*           The prefix to be used for each of the files resulting from the split operation. If no  
 34000       *name* argument is given, 'x' shall be used as the prefix of the output files. The  
 34001       combined length of the basename of *prefix* and *suffix\_length* cannot exceed  
 34002       {NAME\_MAX} bytes. See the OPTIONS section.



34003 **STDIN**

34004 See the INPUT FILES section.

34005 **INPUT FILES**

34006 Any file can be used as input.

34007 **ENVIRONMENT VARIABLES**34008 The following environment variables shall affect the execution of *split*:

34009 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 34010 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 34011 Internationalization Variables for the precedence of internationalization variables  
 34012 used to determine the values of locale categories.)

34013 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 34014 internationalization variables.

34015 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 34016 characters (for example, single-byte as opposed to multi-byte characters in  
 34017 arguments and input files).

34018 *LC\_MESSAGES*

34019 Determine the locale that should be used to affect the format and contents of  
 34020 diagnostic messages written to standard error.

34021 *XS* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34022 **ASYNCHRONOUS EVENTS**

34023 Default.

34024 **STDOUT**

34025 Not used.

34026 **STDERR**

34027 The standard error shall be used only for diagnostic messages.

34028 **OUTPUT FILES**

34029 The output files contain portions of the original input file; otherwise, unchanged.

34030 **EXTENDED DESCRIPTION**

34031 None.

34032 **EXIT STATUS**

34033 The following exit values shall be returned:

34034 0 Successful completion.

34035 &gt;0 An error occurred.

34036 **CONSEQUENCES OF ERRORS**

34037 Default.

34038 **APPLICATION USAGE**

34039 None.

34040 **EXAMPLES**34041 In the following examples **foo** is a text file that contains 5 000 lines.34042 1. Create five files, **xaa**, **xab**, **xac**, **xad**, and **xae**:34043 `split foo`34044 2. Create five files, but the suffixed portion of the created files consists of three letters, **xaaa**,  
34045 **xaab**, **xaac**, **xaad**, and **xaae**:34046 `split -a 3 foo`34047 3. Create three files with four-letter suffixes and a supplied prefix, **bar\_aaaa**, **bar\_aaab**, and  
34048 **bar\_aaac**:34049 `split -a 4 -l 2000 foo bar_`34050 4. Create as many files as are necessary to contain at most 20\*1 024 bytes, each with the  
34051 default prefix of **x** and a five-letter suffix:34052 `split -a 5 -b 20k foo`34053 **RATIONALE**34054 The **-b** option was added to provide a mechanism for splitting files other than by lines. While  
34055 most uses of the **-b** option are for transmitting files over networks, some believed it would have  
34056 additional uses.34057 The **-a** option was added to overcome the limitation of being able to create only 676 files.34058 Consideration was given to deleting this utility, using the rationale that the functionality  
34059 provided by this utility is available via the *csplit* utility (see *csplit*). Upon reconsideration of the  
34060 purpose of the User Portability Extension, it was decided to retain both this utility and the *csplit*  
34061 utility because users use both utilities and have historical expectations of their behavior.  
34062 Furthermore, the splitting on byte boundaries in *split* cannot be duplicated with the historical  
34063 *csplit*.34064 The text “*split* shall not delete the files it created with valid suffixes” would normally be  
34065 assumed, but since the related utility, *csplit*, does delete files under some circumstances, the  
34066 historical behavior of *split* is made explicit to avoid misinterpretation.34067 **FUTURE DIRECTIONS**

34068 None.

34069 **SEE ALSO**34070 *csplit*34071 **CHANGE HISTORY**

34072 First released in Issue 2.

34073 **Issue 6**

34074 This utility is marked as part of the User Portability Utilities option.

34075 The APPLICATION USAGE section is added.

34076 The obsolescent SYNOPSIS is removed.

34077 **NAME**

34078 strings — find printable strings in files

34079 **SYNOPSIS**34080 UP strings [-a] [-t *format*] [-n *number*] [*file...*]

34081

34082 **DESCRIPTION**

34083 The *strings* utility shall look for printable strings in regular files and shall write those strings to  
 34084 standard output. A printable string is any sequence of four (by default) or more printable  
 34085 characters terminated by a <newline> or NUL character. Additional implementation-defined  
 34086 strings may be written; see *localedef*.

34087 **OPTIONS**

34088 The *strings* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 34089 12.2, Utility Syntax Guidelines.

34090 The following options shall be supported:

34091 **-a** Scan files in their entirety. If **-a** is not specified, it is implementation-defined what  
 34092 portion of each file is scanned for strings.

34093 **-n *number*** Specify the minimum string length, where the *number* argument is a positive  
 34094 decimal integer. The default shall be 4.

34095 **-t *format*** Write each string preceded by its byte offset from the start of the file. The format  
 34096 shall be dependent on the single character used as the *format* option-argument:

34097     d   The offset shall be written in decimal.

34098     o   The offset shall be written in octal.

34099     x   The offset shall be written in hexadecimal.

34100 **OPERANDS**

34101 The following operand shall be supported:

34102 *file*       A pathname of a regular file to be used as input. If no *file* operand is specified, the  
 34103 *strings* utility shall read from the standard input.

34104 **STDIN**

34105 See the INPUT FILES section.

34106 **INPUT FILES**

34107 The input files named by the utility arguments or the standard input shall be regular files of any  
 34108 format.

34109 **ENVIRONMENT VARIABLES**34110 The following environment variables shall affect the execution of *strings*:

34111 **LANG**       Provide a default value for the internationalization variables that are unset or null.  
 34112 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 34113 Internationalization Variables for the precedence of internationalization variables  
 34114 used to determine the values of locale categories.)

34115 **LC\_ALL**      If set to a non-empty string value, override the values of all the other  
 34116 internationalization variables.

34117 **LC\_CTYPE**    Determine the locale for the interpretation of sequences of bytes of text data as  
 34118 characters (for example, single-byte as opposed to multi-byte characters in  
 34119 arguments and input files) and to identify printable strings.

34120 *LC\_MESSAGES*

34121 Determine the locale that should be used to affect the format and contents of  
34122 diagnostic messages written to standard error.

34123 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

### 34124 ASYNCHRONOUS EVENTS

34125 Default.

### 34126 STDOUT

34127 Strings found shall be written to the standard output, one per line.

34128 When the **-t** option is not specified, the format of the output shall be:

34129 "%s", <string>

34130 With the **-t o** option, the format of the output shall be:

34131 "%o %s", <byte offset>, <string>

34132 With the **-t x** option, the format of the output shall be:

34133 "%x %s", <byte offset>, <string>

34134 With the **-t d** option, the format of the output shall be:

34135 "%d %s", <byte offset>, <string>

### 34136 STDERR

34137 The standard error shall be used only for diagnostic messages.

### 34138 OUTPUT FILES

34139 None.

### 34140 EXTENDED DESCRIPTION

34141 None.

### 34142 EXIT STATUS

34143 The following exit values shall be returned:

34144 0 Successful completion.

34145 >0 An error occurred.

### 34146 CONSEQUENCES OF ERRORS

34147 Default.

### 34148 APPLICATION USAGE

34149 By default the data area (as opposed to the text, “bss”, or header areas) of a binary executable  
34150 file is scanned. Implementations document which areas are scanned.

34151 Some historical implementations do not require NUL or <newline> terminators for strings to  
34152 permit those languages that do not use NUL as a string terminator to have their strings written.

### 34153 EXAMPLES

34154 None.

### 34155 RATIONALE

34156 Apart from rationalizing the option syntax and slight difficulties with object and executable  
34157 binary files, *strings* is specified to match historical practice closely. The **-a** and **-n** options were  
34158 introduced to replace the non-conforming **-** and *-number* options.

34159 The **-o** option historically means different things on different implementations. Some use it to  
34160 mean “*offset* in decimal”, while others use it as “*offset* in octal”. Instead of trying to decide which

34161 way would be least objectionable, the **-t** option was added. It was originally named **-O** to mean  
34162 “offset”, but was changed to **-t** to be consistent with *od*.

34163 The ISO C standard function *isprint()* is restricted to a domain of **unsigned char**. This volume of  
34164 IEEE Std 1003.1-2001 requires implementations to write strings as defined by the current locale.

34165 **FUTURE DIRECTIONS**

34166 None.

34167 **SEE ALSO**

34168 *localedef, nm*

34169 **CHANGE HISTORY**

34170 First released in Issue 4.

34171 **Issue 6**

34172 This utility is marked as part of the User Portability Utilities option.

34173 The obsolescent SYNOPSIS is removed.

34174 The normative text is reworded to avoid use of the term “must” for application requirements.

34175 **NAME**

34176 strip — remove unnecessary information from executable files (**DEVELOPMENT**)

34177 **SYNOPSIS**

34178 SD strip *file...*

34179

34180 **DESCRIPTION**

34181 The *strip* utility shall remove from executable files named by the *file* operands any information  
 34182 the implementor deems unnecessary for execution of those files. The nature of that information  
 34183 is unspecified. The effect of *strip* shall be similar to the use of the **-s** option to *c99* or *fort77*.

34184 **OPTIONS**

34185 None.

34186 **OPERANDS**

34187 The following operand shall be supported:

34188 *file* A pathname referring to an executable file.

34189 **STDIN**

34190 Not used.

34191 **INPUT FILES**

34192 The input files shall be in the form of executable files successfully produced by any compiler  
 34193 defined by this volume of IEEE Std 1003.1-2001.

34194 **ENVIRONMENT VARIABLES**

34195 The following environment variables shall affect the execution of *strip*:

34196 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 34197 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 34198 Internationalization Variables for the precedence of internationalization variables  
 34199 used to determine the values of locale categories.)

34200 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 34201 internationalization variables.

34202 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 34203 characters (for example, single-byte as opposed to multi-byte characters in  
 34204 arguments).

34205 *LC\_MESSAGES*

34206 Determine the locale that should be used to affect the format and contents of  
 34207 diagnostic messages written to standard error.

34208 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34209 **ASYNCHRONOUS EVENTS**

34210 Default.

34211 **STDOUT**

34212 Not used.

34213 **STDERR**

34214 The standard error shall be used only for diagnostic messages.

34215 **OUTPUT FILES**

34216           The *strip* utility shall produce executable files of unspecified format.

34217 **EXTENDED DESCRIPTION**

34218           None.

34219 **EXIT STATUS**

34220           The following exit values shall be returned:

34221           0   Successful completion.

34222           >0  An error occurred.

34223 **CONSEQUENCES OF ERRORS**

34224           Default.

34225 **APPLICATION USAGE**

34226           None.

34227 **EXAMPLES**

34228           None.

34229 **RATIONALE**

34230           Historically, this utility has been used to remove the symbol table from an executable file. It was  
34231           included since it is known that the amount of symbolic information can amount to several  
34232           megabytes; the ability to remove it in a portable manner was deemed important, especially for  
34233           smaller systems.

34234           The behavior of *strip* is said to be the same as the **-s** option to a compiler. While the end result is  
34235           essentially the same, it is not required to be identical.

34236 **FUTURE DIRECTIONS**

34237           None.

34238 **SEE ALSO**

34239           *ar*, *c99*, *fort77*

34240 **CHANGE HISTORY**

34241           First released in Issue 2.

34242 **Issue 6**

34243           This utility is marked as part of the Software Development Utilities option.

34244 **NAME**

34245           stty — set the options for a terminal

34246 **SYNOPSIS**

34247           stty [ -a | -g ]

34248           stty *operands*34249 **DESCRIPTION**

34250       The *stty* utility shall set or report on terminal I/O characteristics for the device that is its  
 34251       standard input. Without options or operands specified, it shall report the settings of certain  
 34252       characteristics, usually those that differ from implementation-defined defaults. Otherwise, it  
 34253       shall modify the terminal state according to the specified operands. Detailed information about  
 34254       the modes listed in the first five groups below are described in the Base Definitions volume of  
 34255       IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface. Operands in the Combination  
 34256       Modes group (see **Combination Modes** (on page 889)) are implemented using operands in the  
 34257       previous groups. Some combinations of operands are mutually-exclusive on some terminal  
 34258       types; the results of using such combinations are unspecified.

34259       Typical implementations of this utility require a communications line configured to use the  
 34260       **termios** interface defined in the System Interfaces volume of IEEE Std 1003.1-2001. On systems  
 34261       where none of these lines are available, and on lines not currently configured to support the  
 34262       **termios** interface, some of the operands need not affect terminal characteristics.

34263 **OPTIONS**

34264       The *stty* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 34265       12.2, Utility Syntax Guidelines.

34266       The following options shall be supported:

34267       **-a**           Write to standard output all the current settings for the terminal.

34268       **-g**           Write to standard output all the current settings in an unspecified form that can be  
 34269       used as arguments to another invocation of the *stty* utility on the same system. The  
 34270       form used shall not contain any characters that would require quoting to avoid  
 34271       word expansion by the shell; see Section 2.6 (on page 36).

34272 **OPERANDS**

34273       The following operands shall be supported to set the terminal characteristics.

34274       **Control Modes**

34275       **parenb** (**-parenb**) Enable (disable) parity generation and detection. This shall have the effect of  
 34276       setting (not setting) PARENB in the **termios** *c\_flag* field, as defined in the  
 34277       Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General  
 34278       Terminal Interface.

34279       **parodd** (**-parodd**)  
 34280       Select odd (even) parity. This shall have the effect of setting (not setting)  
 34281       PARODD in the **termios** *c\_flag* field, as defined in the Base Definitions  
 34282       volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.

34283       **cs5 cs6 cs7 cs8** Select character size, if possible. This shall have the effect of setting CS5, CS6,  
 34284       CS7, and CS8, respectively, in the **termios** *c\_flag* field, as defined in the Base  
 34285       Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal  
 34286       Interface.

34287       *number*       Set terminal baud rate to the number given, if possible. If the baud rate is set  
 34288       to zero, the modem control lines shall no longer be asserted. This shall have



|       |                                                                                                        |                                                                                                 |
|-------|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| 34289 |                                                                                                        | the effect of setting the input and output <b>termios</b> baud rate values as defined           |
| 34290 |                                                                                                        | in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General                     |
| 34291 |                                                                                                        | Terminal Interface.                                                                             |
| 34292 | <b>ispeed</b> <i>number</i>                                                                            | Set terminal input baud rate to the number given, if possible. If the input baud                |
| 34293 |                                                                                                        | rate is set to zero, the input baud rate shall be specified by the value of the                 |
| 34294 |                                                                                                        | output baud rate. This shall have the effect of setting the input <b>termios</b> baud           |
| 34295 |                                                                                                        | rate values as defined in the Base Definitions volume of IEEE Std 1003.1-2001,                  |
| 34296 |                                                                                                        | Chapter 11, General Terminal Interface.                                                         |
| 34297 | <b>ospeed</b> <i>number</i>                                                                            | Set terminal output baud rate to the number given, if possible. If the output                   |
| 34298 |                                                                                                        | baud rate is set to zero, the modem control lines shall no longer be asserted.                  |
| 34299 |                                                                                                        | This shall have the effect of setting the output <b>termios</b> baud rate values as             |
| 34300 |                                                                                                        | defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11,                     |
| 34301 |                                                                                                        | General Terminal Interface.                                                                     |
| 34302 | <b>hupcl</b> ( <b>-hupcl</b> )                                                                         | Stop asserting modem control lines (do not stop asserting modem control                         |
| 34303 |                                                                                                        | lines) on last close. This shall have the effect of setting (not setting) HUPCL in              |
| 34304 |                                                                                                        | the <b>termios</b> <i>c_cflag</i> field, as defined in the Base Definitions volume of           |
| 34305 |                                                                                                        | IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                   |
| 34306 | <b>hup</b> ( <b>-hup</b> )                                                                             | Equivalent to <b>hupcl</b> ( <b>-hupcl</b> ).                                                   |
| 34307 | <b>cstopb</b> ( <b>-cstopb</b> )                                                                       | Use two (one) stop bits per character. This shall have the effect of setting (not               |
| 34308 |                                                                                                        | setting) CSTOPB in the <b>termios</b> <i>c_cflag</i> field, as defined in the Base Definitions  |
| 34309 |                                                                                                        | volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                         |
| 34310 | <b>cread</b> ( <b>-cread</b> )                                                                         | Enable (disable) the receiver. This shall have the effect of setting (not setting)              |
| 34311 |                                                                                                        | CREAD in the <b>termios</b> <i>c_cflag</i> field, as defined in the Base Definitions volume     |
| 34312 |                                                                                                        | of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                |
| 34313 | <b>clocal</b> ( <b>-clocal</b> )                                                                       | Assume a line without (with) modem control. This shall have the effect of                       |
| 34314 |                                                                                                        | setting (not setting) CLOCAL in the <b>termios</b> <i>c_cflag</i> field, as defined in the      |
| 34315 |                                                                                                        | Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General                            |
| 34316 |                                                                                                        | Terminal Interface.                                                                             |
| 34317 | It is unspecified whether <i>stty</i> shall report an error if an attempt to set a Control Mode fails. |                                                                                                 |
| 34318 | <b>Input Modes</b>                                                                                     |                                                                                                 |
| 34319 | <b>ignbrk</b> ( <b>-ignbrk</b> )                                                                       | Ignore (do not ignore) break on input. This shall have the effect of setting (not               |
| 34320 |                                                                                                        | setting) IGNBRK in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions  |
| 34321 |                                                                                                        | volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                         |
| 34322 | <b>brkint</b> ( <b>-brkint</b> )                                                                       | Signal (do not signal) INTR on break. This shall have the effect of setting (not                |
| 34323 |                                                                                                        | setting) BRKINT in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions  |
| 34324 |                                                                                                        | volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                         |
| 34325 | <b>ignpar</b> ( <b>-ignpar</b> )                                                                       | Ignore (do not ignore) bytes with parity errors. This shall have the effect of                  |
| 34326 |                                                                                                        | setting (not setting) IGNPAR in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base |
| 34327 |                                                                                                        | Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal                        |
| 34328 |                                                                                                        | Interface.                                                                                      |
| 34329 | <b>parmrk</b> ( <b>-parmrk</b> )                                                                       |                                                                                                 |
| 34330 |                                                                                                        | Mark (do not mark) parity errors. This shall have the effect of setting (not                    |
| 34331 |                                                                                                        | setting) PARMRK in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base              |
| 34332 |                                                                                                        | Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal                        |
| 34333 |                                                                                                        | Interface.                                                                                      |

|           |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
|-----------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 34334     | <b>inpck</b> ( <b>-inpck</b> )   | Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                                |
| 34335     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34336     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34337     | <b>istrip</b> ( <b>-istrip</b> ) | Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                  |
| 34338     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34339     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34340     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34341     | <b>inlcr</b> ( <b>-inlcr</b> )   | Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                                    |
| 34342     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34343     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34344     | <b>igncr</b> ( <b>-igncr</b> )   | Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                                    |
| 34345     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34346     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34347     | <b>icrnl</b> ( <b>-icrnl</b> )   | Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                                    |
| 34348     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34349     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34350     | <b>ixon</b> ( <b>-ixon</b> )     | Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface. |
| 34351     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34352     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34353     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34354     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34355 XSI | <b>ixany</b> ( <b>-ixany</b> )   | Allow any character to restart output. This shall have the effect of setting (not setting) IXANY in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                                 |
| 34356     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34357     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34358     | <b>ixoff</b> ( <b>-ixoff</b> )   | Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.          |
| 34359     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34360     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34361     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34362     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34363     | <b>Output Modes</b>              |                                                                                                                                                                                                                                                                                                                                                                         |
| 34364     | <b>opost</b> ( <b>-opost</b> )   | Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                       |
| 34365     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34366     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34367     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34368 XSI | <b>ocrnl</b> ( <b>-ocrnl</b> )   | Map (do not map) CR to NL on output. This shall have the effect of setting (not setting) OCRNL in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                                   |
| 34369     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34370     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34371     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34372     | <b>onocr</b> ( <b>-onocr</b> )   | Do not (do) output CR at column zero. This shall have the effect of setting (not setting) ONOCR in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                                  |
| 34373     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34374     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34375     | <b>onlret</b> ( <b>-onlret</b> ) | The terminal newline key performs (does not perform) the CR function. This shall have the effect of setting (not setting) ONLRET in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                 |
| 34376     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34377     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34378     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |

|       |                                    |                                                                                                                                                                                                                                                                                                                                                               |
|-------|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 34379 | <b>ofill</b> ( <b>-ofill</b> )     | Use fill characters (use timing) for delays. This shall have the effect of setting (not setting) OFILL in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                 |
| 34380 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34381 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34382 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34383 | <b>ofdel</b> ( <b>-ofdel</b> )     | Fill characters are DELs (NULs). This shall have the effect of setting (not setting) OFDEL in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                             |
| 34384 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34385 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34386 | <b>cr0 cr1 cr2 cr3</b>             | Select the style of delay for CRs. This shall have the effect of setting CRDLY to CR0, CR1, CR2, or CR3, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                 |
| 34387 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34388 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34389 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34390 | <b>nl0 nl1</b>                     | Select the style of delay for NL. This shall have the effect of setting NLDLY to NL0 or NL1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                             |
| 34391 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34392 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34393 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34394 | <b>tab0 tab1 tab2 tab3</b>         | Select the style of delay for horizontal tabs. This shall have the effect of setting TABDLY to TAB0, TAB1, TAB2, or TAB3, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface. Note that TAB3 has the effect of expanding <tab>s to <space>s. |
| 34395 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34396 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34397 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34398 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34399 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34400 | <b>tabs</b> ( <b>-tabs</b> )       | Synonym for <b>tab0</b> ( <b>tab3</b> ).                                                                                                                                                                                                                                                                                                                      |
| 34401 | <b>bs0 bs1</b>                     | Select the style of delay for backspaces. This shall have the effect of setting BSDLY to BS0 or BS1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                     |
| 34402 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34403 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34404 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34405 | <b>ff0 ff1</b>                     | Select the style of delay for form-feeds. This shall have the effect of setting FFDLY to FF0 or FF1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                     |
| 34406 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34407 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34408 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34409 | <b>vt0 vt1</b>                     | Select the style of delay for vertical-tabs. This shall have the effect of setting VTDLY to VT0 or VT1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                  |
| 34410 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34411 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34412 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34413 | <b>Local Modes</b>                 |                                                                                                                                                                                                                                                                                                                                                               |
| 34414 | <b>isig</b> ( <b>-isig</b> )       | Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SUSP. This shall have the effect of setting (not setting) ISIG in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                      |
| 34415 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34416 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34417 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34418 | <b>icanon</b> ( <b>-icanon</b> )   | Enable (disable) canonical input (ERASE and KILL processing). This shall have the effect of setting (not setting) ICANON in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                               |
| 34419 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34420 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34421 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34422 | <b>ixexten</b> ( <b>-ixexten</b> ) | Enable (disable) any implementation-defined special control characters not currently controlled by <b>icanon</b> , <b>isig</b> , <b>ixon</b> , or <b>ixoff</b> . This shall have the effect of setting (not setting) IEXTEN in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base                                                                |
| 34423 |                                    |                                                                                                                                                                                                                                                                                                                                                               |
| 34424 |                                    |                                                                                                                                                                                                                                                                                                                                                               |

|       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 34425 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                                                                                                                                                                                         |
| 34426 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34427 | <b>echo</b> ( <b>-echo</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Echo back (do not echo back) every character typed. This shall have the effect of setting (not setting) ECHO in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                         |
| 34428 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34429 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34430 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34431 | <b>echoe</b> ( <b>-echoe</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | The ERASE character visually erases (does not erase) the last character in the current line from the display, if possible. This shall have the effect of setting (not setting) ECHOE in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface. |
| 34432 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34433 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34434 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34435 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34436 | <b>echok</b> ( <b>-echok</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Echo (do not echo) NL after KILL character. This shall have the effect of setting (not setting) ECHOK in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                |
| 34437 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34438 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34439 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34440 | <b>echonl</b> ( <b>-echonl</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Echo (do not echo) NL, even if <b>echo</b> is disabled. This shall have the effect of setting (not setting) ECHONL in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                   |
| 34441 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34442 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34443 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34444 | <b>noflsh</b> ( <b>-noflsh</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of setting (not setting) NOFLSH in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                            |
| 34445 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34446 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34447 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34448 | <b>tostop</b> ( <b>-tostop</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Send SIGTTOU for background output. This shall have the effect of setting (not setting) TOSTOP in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface.                                                                                       |
| 34449 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34450 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34451 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34452 | <b>Special Control Character Assignments</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                                             |
| 34453 | <control>-character string                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                             |
| 34454 | Set <control>-character to <i>string</i> . If <control>-character is one of the character sequences in the first column of the following table, the corresponding Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface control character from the second column shall be recognized. This has the effect of setting the corresponding element of the <b>termios</b> <i>c_cc</i> array (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers, < <b>termios.h</b> >). |                                                                                                                                                                                                                                                                                                                                             |
| 34455 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34456 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34457 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34458 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |
| 34459 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                             |

**Table 4-19** Control Character Names in *stty*

| Control Character | c_cc Subscript | Description     |
|-------------------|----------------|-----------------|
| <b>eof</b>        | VEOF           | EOF character   |
| <b>eol</b>        | VEOL           | EOL character   |
| <b>erase</b>      | VERASE         | ERASE character |
| <b>intr</b>       | VINTR          | INTR character  |
| <b>kill</b>       | VKILL          | KILL character  |
| <b>quit</b>       | VQUIT          | QUIT character  |
| <b>susp</b>       | VSUSP          | SUSP character  |
| <b>start</b>      | VSTART         | START character |
| <b>stop</b>       | VSTOP          | STOP character  |

If *string* is a single character, the control character shall be set to that character. If *string* is the two-character sequence "^-" or the string *undef*, the control character shall be set to `_POSIX_VDISABLE`, if it is in effect for the device; if `_POSIX_VDISABLE` is not in effect for the device, it shall be treated as an error. In the POSIX locale, if *string* is a two-character sequence beginning with circumflex ('^'), and the second character is one of those listed in the "^c" column of the following table, the control character shall be set to the corresponding character value in the Value column of the table.

**Table 4-20** Circumflex Control Characters in *stty*

| ^c   | Value | ^c   | Value | ^c   | Value |
|------|-------|------|-------|------|-------|
| a, A | <SOH> | l, L | <FF>  | w, W | <ETB> |
| b, B | <STX> | m, M | <CR>  | x, X | <CAN> |
| c, C | <ETX> | n, N | <SO>  | y, Y | <EM>  |
| d, D | <EOT> | o, O | <SI>  | z, Z | <SUB> |
| e, E | <ENQ> | p, P | <DLE> | [    | <ESC> |
| f, F | <ACK> | q, Q | <DC1> | \    | <FS>  |
| g, G | <BEL> | r, R | <DC2> | ]    | <GS>  |
| h, H | <BS>  | s, S | <DC3> | ^    | <RS>  |
| i, I | <HT>  | t, T | <DC4> | _    | <US>  |
| j, J | <LF>  | u, U | <NAK> | ?    | <DEL> |
| k, K | <VT>  | v, V | <SYN> |      |       |

**min** *number*

Set the value of MIN to *number*. MIN is used in non-canonical mode input processing (**icanon**).

**time** *number*

Set the value of TIME to *number*. TIME is used in non-canonical mode input processing (**icanon**).

**Combination Modes***saved settings*

Set the current terminal characteristics to the saved settings produced by the **-g** option.

**evenp** or **parity**

Enable **parenb** and **cs7**; disable **parodd**.

**oddp**

Enable **parenb**, **cs7**, and **parodd**.

34504       **–parity, –evenp, or –oddp**  
 34505           Disable **parenb**, and set **cs8**.

34506 xSI   **raw (–raw or cooked)**  
 34507           Enable (disable) raw input and output. Raw mode shall be equivalent to setting:

34508           stty cs8 erase ^- kill ^- intr ^- \  
 34509           quit ^- eof ^- eol ^- –post –inpck

34510       **nl (–nl)**  
 34511           Disable (enable) **icrnl**. In addition, **–nl** unsets **inlcr** and **igncr**.

34512       **ek**   Reset ERASE and KILL characters back to system defaults.

34513       **sane**  
 34514           Reset all modes to some reasonable, unspecified, values.

34515   **STDIN**  
 34516           Although no input is read from standard input, standard input shall be used to get the current  
 34517           terminal I/O characteristics and to set new terminal I/O characteristics.

34518   **INPUT FILES**  
 34519           None.

34520   **ENVIRONMENT VARIABLES**  
 34521           The following environment variables shall affect the execution of *stty*:

34522       **LANG**       Provide a default value for the internationalization variables that are unset or null.  
 34523                      (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 34524                      Internationalization Variables for the precedence of internationalization variables  
 34525                      used to determine the values of locale categories.)

34526       **LC\_ALL**     If set to a non-empty string value, override the values of all the other  
 34527                      internationalization variables.

34528       **LC\_CTYPE**   This variable determines the locale for the interpretation of sequences of bytes of  
 34529                      text data as characters (for example, single-byte as opposed to multi-byte  
 34530                      characters in arguments) and which characters are in the class **print**.

34531       **LC\_MESSAGES**  
 34532                      Determine the locale that should be used to affect the format and contents of  
 34533                      diagnostic messages written to standard error.

34534 xSI   **NLSPATH**   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34535   **ASYNCHRONOUS EVENTS**  
 34536           Default.

34537   **STDOUT**  
 34538           If operands are specified, no output shall be produced.

34539           If the **–g** option is specified, *stty* shall write to standard output the current settings in a form that  
 34540           can be used as arguments to another instance of *stty* on the same system.

34541           If the **–a** option is specified, all of the information as described in the OPERANDS section shall  
 34542           be written to standard output. Unless otherwise specified, this information shall be written as  
 34543           <space>-separated tokens in an unspecified format, on one or more lines, with an unspecified  
 34544           number of tokens per line. Additional information may be written.

34545           If no options or operands are specified, an unspecified subset of the information written for the  
 34546           **–a** option shall be written.

34547 If speed information is written as part of the default output, or if the **-a** option is specified and if  
 34548 the terminal input speed and output speed are the same, the speed information shall be written  
 34549 as follows:

34550 "speed %d baud;", <speed>

34551 Otherwise, speeds shall be written as:

34552 "ispeed %d baud; ospeed %d baud;", <ispeed>, <ospeed>

34553 In locales other than the POSIX locale, the word **baud** may be changed to something more  
 34554 appropriate in those locales.

34555 If control characters are written as part of the default output, or if the **-a** option is specified,  
 34556 control characters shall be written as:

34557 "%s = %s;", <control-character name>, <value>

34558 where <value> is either the character, or some visual representation of the character if it is non-  
 34559 printable, or the string *undef* if the character is disabled.

#### 34560 **STDERR**

34561 The standard error shall be used only for diagnostic messages.

#### 34562 **OUTPUT FILES**

34563 None.

#### 34564 **EXTENDED DESCRIPTION**

34565 None.

#### 34566 **EXIT STATUS**

34567 The following exit values shall be returned:

34568 0 The terminal options were read or set successfully.

34569 >0 An error occurred.

#### 34570 **CONSEQUENCES OF ERRORS**

34571 Default.

#### 34572 **APPLICATION USAGE**

34573 The **-g** flag is designed to facilitate the saving and restoring of terminal state from the shell level.  
 34574 For example, a program may:

```
34575 saveterm="$(stty -g)" # save terminal state
34576 stty (new settings) # set new state
34577 ... # ...
34578 stty $saveterm # restore terminal state
```

34579 Since the format is unspecified, the saved value is not portable across systems.

34580 Since the **-a** format is so loosely specified, scripts that save and restore terminal settings should  
 34581 use the **-g** option.

#### 34582 **EXAMPLES**

34583 None.

#### 34584 **RATIONALE**

34585 The original *stty* description was taken directly from System V and reflected the System V  
 34586 terminal driver **termio**. It has been modified to correspond to the terminal driver **termios**.

34587 Output modes are specified only for XSI-conformant systems. All implementations are expected  
 34588 to provide *stty* operands corresponding to all of the output modes they support.

34589 The *stty* utility is primarily used to tailor the user interface of the terminal, such as selecting the  
 34590 preferred ERASE and KILL characters. As an application programming utility, *stty* can be used  
 34591 within shell scripts to alter the terminal settings for the duration of the script.

34592 The **termios** section states that individual disabling of control characters is possible through the  
 34593 option `_POSIX_VDISABLE`. If enabled, two conventions currently exist for specifying this:  
 34594 System V uses `^_`, and BSD uses *undef*. Both are accepted by *stty* in this volume of  
 34595 IEEE Std 1003.1-2001. The other BSD convention of using the letter `'u'` was rejected because it  
 34596 conflicts with the actual letter `'u'`, which is an acceptable value for a control character.

34597 Early proposals did not specify the mapping of `^c` to control characters because the control  
 34598 characters were not specified in the POSIX locale character set description file requirements. The  
 34599 control character set is now specified in the Base Definitions volume of IEEE Std 1003.1-2001,  
 34600 Chapter 3, Definitions so the historical mapping is specified. Note that although the mapping  
 34601 corresponds to control-character key assignments on many terminals that use the  
 34602 ISO/IEC 646:1991 standard (or ASCII) character encodings, the mapping specified here is to the  
 34603 control characters, not their keyboard encodings.

34604 Since **termios** supports separate speeds for input and output, two new options were added to  
 34605 specify each distinctly.

34606 Some historical implementations use standard input to get and set terminal characteristics;  
 34607 others use standard output. Since input from a login TTY is usually restricted to the owner while  
 34608 output to a TTY is frequently open to anyone, using standard input provides fewer chances of  
 34609 accidentally (or maliciously) altering the terminal settings of other users. Using standard input  
 34610 also allows *stty -a* and *stty -g* output to be redirected for later use. Therefore, usage of standard  
 34611 input is required by this volume of IEEE Std 1003.1-2001.

#### 34612 **FUTURE DIRECTIONS**

34613 None.

#### 34614 **SEE ALSO**

34615 Chapter 2 (on page 29), the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11,  
 34616 General Terminal Interface, **<termios.h>**

#### 34617 **CHANGE HISTORY**

34618 First released in Issue 2.

#### 34619 **Issue 5**

34620 The description of **tabs** is clarified.

34621 The FUTURE DIRECTIONS section is added.

#### 34622 **Issue 6**

34623 The legacy items **iucl**(**-iucl**), **xcas**, **olcu**(**-olcu**), **lcas**(**-lcas**), and **LCASE**(**-LCASE**) are  
 34624 removed.

34625 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/37 is applied, applying IEEE PASC  
 34626 Interpretation 1003.2 #133, fixing an error in the OPERANDS section for the Combination Modes  
 34627 **nl**(**-nl**).



34628 **NAME**34629            `tabs` — set terminal tabs34630 **SYNOPSIS**34631 UP XSI    `tabs [ -n | -a | -a2 | -c | -c2 | -c3 | -f | -p | -s | -u ] [+m[n]] [-T type]`34632            `tabs [-T type] [ +[n]] n1[,n2,...]`

34633

34634 **DESCRIPTION**

34635            The `tabs` utility shall display a series of characters that first clears the hardware terminal tab  
 34636 XSI        settings and then initializes the tab stops at the specified positions and optionally adjusts the  
 34637            margin.

34638            The phrase “tab-stop position *N*” shall be taken to mean that, from the start of a line of output,  
 34639            tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position  
 34640            on that line. The maximum number of tab stops allowed is terminal-dependent.

34641            It need not be possible to implement `tabs` on certain terminals. If the terminal type obtained from  
 34642            the `TERM` environment variable or `-T` option represents such a terminal, an appropriate  
 34643            diagnostic message shall be written to standard error and `tabs` shall exit with a status greater  
 34644            than zero.

34645 **OPTIONS**

34646            The `tabs` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 34647 XSI        12.2, Utility Syntax Guidelines, except for various extensions: the options `-a2`, `-c2`, and `-c3` are  
 34648            multi-character.

34649            The following options shall be supported:

34650            `-n`            Specify repetitive tab stops separated by a uniform number of column positions, *n*,  
 34651            where *n* is a single-digit decimal number. The default usage of `tabs` with no  
 34652            arguments shall be equivalent to `tabs-8`. When `-0` is used, the tab stops shall be  
 34653            cleared and no new ones set.

34654 XSI        `-a`            1,10,16,36,72  
 34655            Assembler, applicable to some mainframes.

34656 XSI        `-a2`            1,10,16,40,72  
 34657            Assembler, applicable to some mainframes.

34658 XSI        `-c`            1,8,12,16,20,55  
 34659            COBOL, normal format.

34660 XSI        `-c2`            1,6,10,14,49  
 34661            COBOL, compact format (columns 1 to 6 omitted).

34662 XSI        `-c3`            1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67  
 34663            COBOL compact format (columns 1 to 6 omitted), with more tabs than `-c2`.

34664 XSI        `-f`            1,7,11,15,19,23  
 34665            FORTRAN

34666 XSI        `-p`            1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61  
 34667            PL / 1

34668 XSI        `-s`            1,10,55  
 34669            SNOBOL

34670 XSI        `-u`            1,12,20,44  
 34671            Assembler, applicable to some mainframes.

34672        **-T type**       Indicate the type of terminal. If this option is not supplied and the *TERM* variable  
34673                           is unset or null, an unspecified default terminal type shall be used. The setting of  
34674                           *type* shall take precedence over the value in *TERM*.

#### 34675 **OPERANDS**

34676        The following operand shall be supported:

34677        **n1[,n2,...]**     A single command line argument that consists of tab-stop values separated using  
34678                           either commas or <blank>s. The application shall ensure that the tab-stop values  
34679                           are positive decimal integers in strictly ascending order. If any number (except the  
34680                           first one) is preceded by a plus sign, it is taken as an increment to be added to the  
34681                           previous value. For example, the tab lists 1,10,20,30 and 1,10,+10,+10 are  
34682                           considered to be identical.

#### 34683 **STDIN**

34684        Not used.

#### 34685 **INPUT FILES**

34686        None.

#### 34687 **ENVIRONMENT VARIABLES**

34688        The following environment variables shall affect the execution of *tabs*:

34689        **LANG**           Provide a default value for the internationalization variables that are unset or null.  
34690                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
34691                           Internationalization Variables for the precedence of internationalization variables  
34692                           used to determine the values of locale categories.)

34693        **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
34694                           internationalization variables.

34695        **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
34696                           characters (for example, single-byte as opposed to multi-byte characters in  
34697                           arguments).

34698        **LC\_MESSAGES**

34699                           Determine the locale that should be used to affect the format and contents of  
34700                           diagnostic messages written to standard error.

34701 xSI     **NLSPATH**     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34702        **TERM**           Determine the terminal type. If this variable is unset or null, and if the **-T** option is  
34703                           not specified, an unspecified default terminal type shall be used.

#### 34704 **ASYNCHRONOUS EVENTS**

34705        Default.

#### 34706 **STDOUT**

34707        If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be  
34708        written to standard output in an unspecified format. If standard output is not a terminal,  
34709        undefined results occur.

#### 34710 **STDERR**

34711        The standard error shall be used only for diagnostic messages.

#### 34712 **OUTPUT FILES**

34713        None.

34714 **EXTENDED DESCRIPTION**

34715 None.

34716 **EXIT STATUS**

34717 The following exit values shall be returned:

34718 0 Successful completion.

34719 &gt;0 An error occurred.

34720 **CONSEQUENCES OF ERRORS**

34721 Default.

34722 **APPLICATION USAGE**34723 This utility makes use of the terminal's hardware tabs and the *stty tabs* option.

34724 This utility is not recommended for application use.

34725 Some integrated display units might not have escape sequences to set tab stops, but may be set  
 34726 by internal system calls. On these terminals, *tabs* works if standard output is directed to the  
 34727 terminal; if output is directed to another file, however, *tabs* fails.

34728 **EXAMPLES**

34729 None.

34730 **RATIONALE**

34731 Consideration was given to having the *tput* utility handle all of the functions described in *tabs*.  
 34732 However, the separate *tabs* utility was retained because it seems more intuitive to use a  
 34733 command named *tabs* than *tput* with a new option. The *tput* utility does not support setting or  
 34734 clearing tabs, and no known historical version of *tabs* supports the capability of setting arbitrary  
 34735 tab stops.

34736 The System V *tabs* interface is very complex; the version in this volume of IEEE Std 1003.1-2001  
 34737 has a reduced feature list, but many of the features omitted were restored as XSI extensions even  
 34738 though the supported languages and coding styles are primarily historical.

34739 There was considerable sentiment for specifying only a means of resetting the tabs back to a  
 34740 known state—presumably the “standard” of tabs every eight positions. The following features  
 34741 were omitted:

- 34742 • Setting tab stops via the first line in a file, using *--file*. Since even the SVID has no complete  
 34743 explanation of this feature, it is doubtful that it is in widespread use.

34744 In an early proposal, a *-t tablist* option was added for consistency with *expand*; this was later  
 34745 removed when inconsistencies with the historical list of tabs were identified.

34746 Consideration was given to adding a *-p* option that would output the current tab settings so  
 34747 that they could be saved and then later restored. This was not accepted because querying the tab  
 34748 stops of the terminal is not a capability in historical *terminfo* or *termcap* facilities and might not be  
 34749 supported on a wide range of terminals.

34750 **FUTURE DIRECTIONS**

34751 None.

34752 **SEE ALSO**34753 *expand*, *stty*, *tput*, *unexpand*

34754 **CHANGE HISTORY**

34755 First released in Issue 2.

34756 **Issue 6**

34757 This utility is marked as part of the User Portability Utilities option.

34758 The normative text is reworded to avoid use of the term “must” for application requirements.

34759 **NAME**34760 `tail` — copy the last part of a file34761 **SYNOPSIS**34762 `tail [-f] [ -c number | -n number] [file]`34763 **DESCRIPTION**34764 The *tail* utility shall copy its input file to the standard output beginning at a designated place.

34765 Copying shall begin at the point in the file indicated by the `-c number` or `-n number` options. The  
 34766 option-argument *number* shall be counted in units of lines or bytes, according to the options `-n`  
 34767 and `-c`. Both line and byte counts start from 1.

34768 Tails relative to the end of the file may be saved in an internal buffer, and thus may be limited in  
 34769 length. Such a buffer, if any, shall be no smaller than {LINE\_MAX}\*10 bytes.

34770 **OPTIONS**

34771 The *tail* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 34772 12.2, Utility Syntax Guidelines.

34773 The following options shall be supported:

34774 `-c number` The application shall ensure that the *number* option-argument is a decimal integer  
 34775 whose sign affects the location in the file, measured in bytes, to begin the copying:

34776

34777

34778

34779

| Sign        | Copying Starts                         |
|-------------|----------------------------------------|
| +           | Relative to the beginning of the file. |
| -           | Relative to the end of the file.       |
| <i>none</i> | Relative to the end of the file.       |

34780 The origin for counting shall be 1; that is, `-c +1` represents the first byte of the file,  
 34781 `-c -1` the last.

34782 `-f` If the input file is a regular file or if the *file* operand specifies a FIFO, do not  
 34783 terminate after the last line of the input file has been copied, but read and copy  
 34784 further bytes from the input file when they become available. If no *file* operand is  
 34785 specified and standard input is a pipe, the `-f` option shall be ignored. If the input  
 34786 file is not a FIFO, pipe, or regular file, it is unspecified whether or not the `-f` option  
 34787 shall be ignored.

34788 `-n number` This option shall be equivalent to `-c number`, except the starting location in the file  
 34789 shall be measured in lines instead of bytes. The origin for counting shall be 1; that  
 34790 is, `-n +1` represents the first line of the file, `-n -1` the last.

34791 If neither `-c` nor `-n` is specified, `-n 10` shall be assumed.34792 **OPERANDS**

34793 The following operand shall be supported:

34794 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
 34795 shall be used.

34796 **STDIN**

34797 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 34798 section.

34799 **INPUT FILES**

34800 If the **-c** option is specified, the input file can contain arbitrary data; otherwise, the input file  
 34801 shall be a text file.

34802 **ENVIRONMENT VARIABLES**

34803 The following environment variables shall affect the execution of *tail*:

34804 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 34805 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 34806 Internationalization Variables for the precedence of internationalization variables  
 34807 used to determine the values of locale categories.)

34808 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 34809 internationalization variables.

34810 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 34811 characters (for example, single-byte as opposed to multi-byte characters in  
 34812 arguments and input files).

34813 **LC\_MESSAGES**  
 34814 Determine the locale that should be used to affect the format and contents of  
 34815 diagnostic messages written to standard error.

34816 **XS** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

34817 **ASYNCHRONOUS EVENTS**

34818 Default.

34819 **STDOUT**

34820 The designated portion of the input file shall be written to standard output.

34821 **STDERR**

34822 The standard error shall be used only for diagnostic messages.

34823 **OUTPUT FILES**

34824 None.

34825 **EXTENDED DESCRIPTION**

34826 None.

34827 **EXIT STATUS**

34828 The following exit values shall be returned:

34829 0 Successful completion.

34830 >0 An error occurred.

34831 **CONSEQUENCES OF ERRORS**

34832 Default.

34833 **APPLICATION USAGE**

34834 The **-c** option should be used with caution when the input is a text file containing multi-byte  
 34835 characters; it may produce output that does not start on a character boundary.

34836 Although the input file to *tail* can be any type, the results might not be what would be expected  
 34837 on some character special device files or on file types not described by the System Interfaces  
 34838 volume of IEEE Std 1003.1-2001. Since this volume of IEEE Std 1003.1-2001 does not specify the  
 34839 block size used when doing input, *tail* need not read all of the data from devices that only  
 34840 perform block transfers.

34841 **EXAMPLES**

34842       The **-f** option can be used to monitor the growth of a file that is being written by some other  
34843       process. For example, the command:

34844       tail -f fred

34845       prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between  
34846       the time *tail* is initiated and killed. As another example, the command:

34847       tail -f -c 15 fred

34848       prints the last 15 bytes of the file **fred**, followed by any bytes that are appended to **fred** between  
34849       the time *tail* is initiated and killed.

34850 **RATIONALE**

34851       This version of *tail* was created to allow conformance to the Utility Syntax Guidelines. The  
34852       historical **-b** option was omitted because of the general non-portability of block-sized units of  
34853       text. The **-c** option historically meant “characters”, but this volume of IEEE Std 1003.1-2001  
34854       indicates that it means “bytes”. This was selected to allow reasonable implementations when  
34855       multi-byte characters are possible; it was not named **-b** to avoid confusion with the historical  
34856       **-b**.

34857       The origin of counting both lines and bytes is 1, matching all widespread historical  
34858       implementations.

34859       The restriction on the internal buffer is a compromise between the historical System V  
34860       implementation of 4 096 bytes and the BSD 32 768 bytes.

34861       The **-f** option has been implemented as a loop that sleeps for 1 second and copies any bytes that  
34862       are available. This is sufficient, but if more efficient methods of determining when new data are  
34863       available are developed, implementations are encouraged to use them.

34864       Historical documentation indicates that *tail* ignores the **-f** option if the input file is a pipe (pipe  
34865       and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System  
34866       V-based systems, this was true when input was taken from standard input, but it did not ignore  
34867       the **-f** flag if a FIFO was named as the *file* operand. Since the **-f** option is not useful on pipes and  
34868       all historical implementations ignore **-f** if no *file* operand is specified and standard input is a  
34869       pipe, this volume of IEEE Std 1003.1-2001 requires this behavior. However, since the **-f** option is  
34870       useful on a FIFO, this volume of IEEE Std 1003.1-2001 also requires that if standard input is a  
34871       FIFO or a FIFO is named, the **-f** option shall not be ignored. Although historical behavior does  
34872       not ignore the **-f** option for other file types, this is unspecified so that implementations are  
34873       allowed to ignore the **-f** option if it is known that the file cannot be extended.

34874       This was changed to the current form based on comments noting that **-c** was almost never used  
34875       without specifying a number and that there was no need to specify **-l** if **-n number** was given.

34876 **FUTURE DIRECTIONS**

34877       None.

34878 **SEE ALSO**

34879       *head*

34880 **CHANGE HISTORY**

34881       First released in Issue 2.

34882 **Issue 6**

34883       The obsolescent SYNOPSIS lines and associated text are removed.

34884       The normative text is reworded to avoid use of the term “must” for application requirements.

34885 **NAME**

34886           talk — talk to another user

34887 **SYNOPSIS**34888 UP        talk *address* [*terminal*]

34889

34890 **DESCRIPTION**34891       The *talk* utility is a two-way, screen-oriented communication program.34892       When first invoked, *talk* shall send a message similar to:34893       Message from <*unspecified string*>34894       talk: connection requested by *your\_address*34895       talk: respond with: talk *your\_address*34896       to the specified *address*. At this point, the recipient of the message can reply by typing:34897       talk *your\_address*

34898       Once communication is established, the two parties can type simultaneously, with their output displayed in separate regions of the screen. Characters shall be processed as follows:

- 34900       • Typing the alert character shall alert the recipient's terminal.
- 34901       • Typing <control>-L shall cause the sender's screen regions to be refreshed.
- 34902       • Typing the erase and kill characters shall affect the sender's terminal in the manner described
- 34903       by the **termios** interface in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11,
- 34904       General Terminal Interface.
- 34905       • Typing the interrupt or end-of-file characters shall terminate the local *talk* utility. Once the
- 34906       *talk* session has been terminated on one side, the other side of the *talk* session shall be notified
- 34907       that the *talk* session has been terminated and shall be able to do nothing except exit.
- 34908       • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those characters
- 34909       to be sent to the recipient's terminal.
- 34910       • When and only when the *stty* **ixten** local mode is enabled, the existence and processing of
- 34911       additional special control characters and multi-byte or single-byte functions shall be
- 34912       implementation-defined.
- 34913       • Typing other non-printable characters shall cause implementation-defined sequences of
- 34914       printable characters to be sent to the recipient's terminal.

34915       Permission to be a recipient of a *talk* message can be denied or granted by use of the *mesg* utility.

34916       However, a user's privilege may further constrain the domain of accessibility of other users'

34917       terminals. The *talk* utility shall fail when the user lacks the appropriate privileges to perform the

34918       requested action.

34919       Certain block-mode terminals do not have all the capabilities necessary to support the

34920       simultaneous exchange of messages required for *talk*. When this type of exchange cannot be

34921       supported on such terminals, the implementation may support an exchange with reduced levels

34922       of simultaneous interaction or it may report an error describing the terminal-related deficiency.

34923 **OPTIONS**

34924       None.



34925 **OPERANDS**

34926 The following operands shall be supported:

34927 *address* The recipient of the *talk* session. One form of *address* is the <*user name*>, as returned  
 34928 by the *who* utility. Other address formats and how they are handled are  
 34929 unspecified.

34930 *terminal* If the recipient is logged in more than once, the *terminal* argument can be used to  
 34931 indicate the appropriate terminal name. If *terminal* is not specified, the *talk* message  
 34932 shall be displayed on one or more accessible terminals in use by the recipient. The  
 34933 format of *terminal* shall be the same as that returned by the *who* utility.

34934 **STDIN**

34935 Characters read from standard input shall be copied to the recipient's terminal in an unspecified  
 34936 manner. If standard input is not a terminal, *talk* shall write a diagnostic message and exit with a  
 34937 non-zero status.

34938 **INPUT FILES**

34939 None.

34940 **ENVIRONMENT VARIABLES**34941 The following environment variables shall affect the execution of *talk*:

34942 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 34943 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 34944 Internationalization Variables for the precedence of internationalization variables  
 34945 used to determine the values of locale categories.)

34946 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 34947 internationalization variables.

34948 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 34949 characters (for example, single-byte as opposed to multi-byte characters in  
 34950 arguments and input files). If the recipient's locale does not use an *LC\_CTYPE*  
 34951 equivalent to the sender's, the results are undefined.

34952 *LC\_MESSAGES*

34953 Determine the locale that should be used to affect the format and contents of  
 34954 diagnostic messages written to standard error and informative messages written to  
 34955 standard output.

34956 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34957 *TERM* Determine the name of the invoker's terminal type. If this variable is unset or null,  
 34958 an unspecified default terminal type shall be used.

34959 **ASYNCHRONOUS EVENTS**

34960 When the *talk* utility receives a SIGINT signal, the utility shall terminate and exit with a zero  
 34961 status. It shall take the standard action for all other signals.

34962 **STDOUT**

34963 If standard output is a terminal, characters copied from the recipient's standard input may be  
 34964 written to standard output. Standard output also may be used for diagnostic messages. If  
 34965 standard output is not a terminal, *talk* shall exit with a non-zero status.

34966 **STDERR**

34967 None.

34968 **OUTPUT FILES**

34969 None.

34970 **EXTENDED DESCRIPTION**

34971 None.

34972 **EXIT STATUS**

34973 The following exit values shall be returned:

34974 0 Successful completion.

34975 >0 An error occurred or *talk* was invoked on a terminal incapable of supporting it.34976 **CONSEQUENCES OF ERRORS**

34977 Default.

34978 **APPLICATION USAGE**

34979 Because the handling of non-printable, non-`<space>s` is tied to the *stty* description of **ixtext**,  
 34980 implementation extensions within the terminal driver can be accessed. For example, some  
 34981 implementations provide line editing functions with certain control character sequences.

34982 **EXAMPLES**

34983 None.

34984 **RATIONALE**

34985 The *write* utility was included in this volume of IEEE Std 1003.1-2001 since it can be  
 34986 implemented on all terminal types. The *talk* utility, which cannot be implemented on certain  
 34987 terminals, was considered to be a “better” communications interface. Both of these programs are  
 34988 in widespread use on historical implementations. Therefore, both utilities have been specified.

34989 All references to networking abilities (*talking* to a user on another system) were removed as  
 34990 being outside the scope of this volume of IEEE Std 1003.1-2001.

34991 Historical BSD and System V versions of *talk* terminate both of the conversations when either  
 34992 user breaks out of the session. This can lead to adverse consequences if a user unwittingly  
 34993 continues to enter text that is interpreted by the shell when the other terminates the session.  
 34994 Therefore, the version of *talk* specified by this volume of IEEE Std 1003.1-2001 requires both  
 34995 users to terminate their end of the session explicitly.

34996 Only messages sent to the terminal of the invoking user can be internationalized in any way:

- 34997 • The original “Message from *<unspecified string>* ...” message sent to the terminal of the  
 34998 recipient cannot be internationalized because the environment of the recipient is as yet  
 34999 inaccessible to the *talk* utility. The environment of the invoking party is irrelevant.
- 35000 • Subsequent communication between the two parties cannot be internationalized because the  
 35001 two parties may specify different languages in their environment (and non-portable  
 35002 characters cannot be mapped from one language to another).
- 35003 • Neither party can be required to communicate in a language other than C and/or the one  
 35004 specified by their environment because unavailable terminal hardware support (for example,  
 35005 fonts) may be required.

35006 The text in the STDOUT section reflects the usage of the verb “display” in this section; some *talk*  
 35007 implementations actually use standard output to write to the terminal, but this volume of  
 35008 IEEE Std 1003.1-2001 does not require that to be the case.

35009 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
 35010 require that they all use or accept the same format.

35011 The handling of non-printable characters is partially implementation-defined because the details  
35012 of mapping them to printable sequences is not needed by the user. Historical implementations,  
35013 for security reasons, disallow the transmission of non-printable characters that may send  
35014 commands to the other terminal.

35015 **FUTURE DIRECTIONS**

35016 None.

35017 **SEE ALSO**

35018 *mesg*, *stty*, *who*, *write*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General  
35019 Terminal Interface

35020 **CHANGE HISTORY**

35021 First released in Issue 4.

35022 **Issue 6**

35023 This utility is marked as part of the User Portability Utilities option.

35024 **NAME**

35025           tee — duplicate standard input

35026 **SYNOPSIS**35027           tee [-ai] [*file...*]35028 **DESCRIPTION**35029           The *tee* utility shall copy standard input to standard output, making a copy in zero or more files.35030           The *tee* utility shall not buffer output.35031           If the **-a** option is not specified, output files shall be written (see Section 1.7.1.4 (on page 4)).35032 **OPTIONS**35033           The *tee* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
35034           Utility Syntax Guidelines.

35035           The following options shall be supported:

35036           **-a**           Append the output to the files.35037           **-i**           Ignore the SIGINT signal.35038 **OPERANDS**

35039           The following operands shall be supported:

35040           *file*           A pathname of an output file. Processing of at least 13 *file* operands shall be  
35041           supported.35042 **STDIN**

35043           The standard input can be of any type.

35044 **INPUT FILES**

35045           None.

35046 **ENVIRONMENT VARIABLES**35047           The following environment variables shall affect the execution of *tee*:35048           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
35049           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
35050           Internationalization Variables for the precedence of internationalization variables  
35051           used to determine the values of locale categories.)35052           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
35053           internationalization variables.35054           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
35055           characters (for example, single-byte as opposed to multi-byte characters in  
35056           arguments).35057           **LC\_MESSAGES**35058                               Determine the locale that should be used to affect the format and contents of  
35059           diagnostic messages written to standard error.35060 **XSI**           **NLSPATH**   Determine the location of message catalogs for the processing of **LC\_MESSAGES**.35061 **ASYNCHRONOUS EVENTS**35062           Default, except that if the **-i** option was specified, SIGINT shall be ignored.

**35063 STDOUT**

35064 The standard output shall be a copy of the standard input.

**35065 STDERR**

35066 The standard error shall be used only for diagnostic messages.

**35067 OUTPUT FILES**

35068 If any *file* operands are specified, the standard input shall be copied to each named file.

**35069 EXTENDED DESCRIPTION**

35070 None.

**35071 EXIT STATUS**

35072 The following exit values shall be returned:

35073 0 The standard input was successfully copied to all output files.

35074 >0 An error occurred.

**35075 CONSEQUENCES OF ERRORS**

35076 If a write to any successfully opened *file* operand fails, writes to other successfully opened *file*  
35077 operands and standard output shall continue, but the exit status shall be non-zero. Otherwise,  
35078 the default actions specified in Section 1.11 (on page 20) apply.

**35079 APPLICATION USAGE**

35080 The *tee* utility is usually used in a pipeline, to make a copy of the output of some utility.

35081 The *file* operand is technically optional, but *tee* is no more useful than *cat* when none is specified.

**35082 EXAMPLES**

35083 Save an unsorted intermediate form of the data in a pipeline:

35084 ... | tee unsorted | sort > sorted

**35085 RATIONALE**

35086 The buffering requirement means that *tee* is not allowed to use ISO C standard fully buffered or  
35087 line-buffered writes. It does not mean that *tee* has to do 1-byte reads followed by 1-byte writes.

35088 It should be noted that early versions of BSD ignore any invalid options and accept a single '-'  
35089 as an alternative to -i. They also print a message if unable to open a file:

35090 "tee: cannot access %s\n", <pathname>

35091 Historical implementations ignore write errors. This is explicitly not permitted by this volume of  
35092 IEEE Std 1003.1-2001.

35093 Some historical implementations use O\_APPEND when providing append mode; others use the  
35094 *lseek()* function to seek to the end-of-file after opening the file without O\_APPEND. This volume  
35095 of IEEE Std 1003.1-2001 requires functionality equivalent to using O\_APPEND; see Section  
35096 1.7.1.4 (on page 4).

**35097 FUTURE DIRECTIONS**

35098 None.

**35099 SEE ALSO**

35100 Chapter 1 (on page 1), *cat*, the System Interfaces volume of IEEE Std 1003.1-2001, *lseek()*

**35101 CHANGE HISTORY**

35102 First released in Issue 2.

35103 **Issue 6**

35104 IEEE PASC Interpretation 1003.2 #168 is applied.

35105 **NAME**

35106           test — evaluate expression

35107 **SYNOPSIS**35108           test [*expression*]35109           [ [*expression*] ]35110 **DESCRIPTION**

35111           The *test* utility shall evaluate the *expression* and indicate the result of the evaluation by its exit  
 35112           status. An exit status of zero indicates that the expression evaluated as true and an exit status of  
 35113           1 indicates that the expression evaluated as false.

35114           In the second form of the utility, which uses "[]" rather than *test*, the application shall ensure  
 35115           that the square brackets are separate arguments.

35116 **OPTIONS**

35117           The *test* utility shall not recognize the "--" argument in the manner specified by guideline 10 in  
 35118           the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines.

35119           No options shall be supported.

35120 **OPERANDS**

35121           The application shall ensure that all operators and elements of primaries are presented as  
 35122           separate arguments to the *test* utility.

35123           The following primaries can be used to construct *expression*:35124           **-b file**           True if *file* exists and is a block special file.35125           **-c file**           True if *file* exists and is a character special file.35126           **-d file**           True if *file* exists and is a directory.35127           **-e file**           True if *file* exists.35128           **-f file**           True if *file* exists and is a regular file.35129           **-g file**           True if *file* exists and its set-group-ID flag is set.35130           **-h file**           True if *file* exists and is a symbolic link.35131           **-L file**           True if *file* exists and is a symbolic link.35132           **-n string**       True if the length of *string* is non-zero.35133           **-p file**           True if *file* is a FIFO.

35134           **-r file**           True if *file* exists and is readable. True shall indicate that permission to read from  
 35135           *file* will be granted, as defined in Section 1.7.1.4 (on page 4).

35136           **-S file**           True if *file* exists and is a socket.35137           **-s file**           True if *file* exists and has a size greater than zero.35138           **-t file\_descriptor**

35139           True if the file whose file descriptor number is *file\_descriptor* is open and is  
 35140           associated with a terminal.

35141           **-u file**           True if *file* exists and its set-user-ID flag is set.

35142           **-w file**           True if *file* exists and is writable. True shall indicate that permission to write from  
 35143           *file* will be granted, as defined in Section 1.7.1.4 (on page 4).

35144 **-x file** True if *file* exists and is executable. True shall indicate that permission to execute  
 35145 *file* will be granted, as defined in Section 1.7.1.4 (on page 4). If *file* is a directory,  
 35146 true shall indicate that permission to search *file* will be granted.

35147 **-z string** True if the length of string *string* is zero.

35148 *string* True if the string *string* is not the null string.

35149 *s1 = s2* True if the strings *s1* and *s2* are identical.

35150 *s1 != s2* True if the strings *s1* and *s2* are not identical.

35151 *n1 -eq n2* True if the integers *n1* and *n2* are algebraically equal.

35152 *n1 -ne n2* True if the integers *n1* and *n2* are not algebraically equal.

35153 *n1 -gt n2* True if the integer *n1* is algebraically greater than the integer *n2*.

35154 *n1 -ge n2* True if the integer *n1* is algebraically greater than or equal to the integer *n2*.

35155 *n1 -lt n2* True if the integer *n1* is algebraically less than the integer *n2*.

35156 *n1 -le n2* True if the integer *n1* is algebraically less than or equal to the integer *n2*.

35157 xSI *expression1 -a expression2*  
 35158 True if both *expression1* and *expression2* are true. The **-a** binary primary is left  
 35159 associative. It has a higher precedence than **-o**.

35160 xSI *expression1 -o expression2*  
 35161 True if either *expression1* or *expression2* is true. The **-o** binary primary is left  
 35162 associative.

35163 With the exception of the **-h file** and **-L file** primaries, if a *file* argument is a symbolic link, *test*  
 35164 shall evaluate the expression by resolving the symbolic link and using the file referenced by the  
 35165 link.

35166 These primaries can be combined with the following operators:

35167 **! expression** True if *expression* is false.

35168 xSI **( expression )** True if *expression* is true. The parentheses can be used to alter the normal  
 35169 precedence and associativity.

35170 The primaries with two elements of the form:

35171 *-primary\_operator primary\_operand*

35172 are known as *unary primaries*. The primaries with three elements in either of the two forms:

35173 *primary\_operand -primary\_operator primary\_operand*

35174 *primary\_operand primary\_operator primary\_operand*

35175 are known as *binary primaries*. Additional implementation-defined operators and  
 35176 *primary\_operators* may be provided by implementations. They shall be of the form *-operator*  
 35177 where the first character of *operator* is not a digit.

35178 The algorithm for determining the precedence of the operators and the return value that shall be  
 35179 generated is based on the number of arguments presented to *test*. (However, when using the  
 35180 "[ . . . ]" form, the right-bracket final argument shall not be counted in this algorithm.)

35181 In the following list, \$1, \$2, \$3, and \$4 represent the arguments presented to *test*:

35182 0 arguments: Exit false (1).



|           |                              |                                                                                                                                                       |
|-----------|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| 35183     | 1 argument:                  | Exit true (0) if \$1 is not null; otherwise, exit false.                                                                                              |
| 35184     | 2 arguments:                 | <ul style="list-style-type: none"> <li>• If \$1 is ' ! ', exit true if \$2 is null, false if \$2 is not null.</li> </ul>                              |
| 35185     |                              | <ul style="list-style-type: none"> <li>• If \$1 is a unary primary, exit true if the unary test is true, false if the unary test is false.</li> </ul> |
| 35186     |                              |                                                                                                                                                       |
| 35187     |                              | <ul style="list-style-type: none"> <li>• Otherwise, produce unspecified results.</li> </ul>                                                           |
| 35188     | 3 arguments:                 | <ul style="list-style-type: none"> <li>• If \$2 is a binary primary, perform the binary test of \$1 and \$3.</li> </ul>                               |
| 35189     |                              | <ul style="list-style-type: none"> <li>• If \$1 is ' ! ', negate the two-argument test of \$2 and \$3.</li> </ul>                                     |
| 35190 XSI |                              | <ul style="list-style-type: none"> <li>• If \$1 is ' ( ' and \$3 is ' ) ', perform the unary test of \$2.</li> </ul>                                  |
| 35191     |                              | <ul style="list-style-type: none"> <li>• Otherwise, produce unspecified results.</li> </ul>                                                           |
| 35192     | 4 arguments:                 | <ul style="list-style-type: none"> <li>• If \$1 is ' ! ', negate the three-argument test of \$2, \$3, and \$4.</li> </ul>                             |
| 35193 XSI |                              | <ul style="list-style-type: none"> <li>• If \$1 is ' ( ' and \$4 is ' ) ', perform the two-argument test of \$2 and \$3.</li> </ul>                   |
| 35194     |                              | <ul style="list-style-type: none"> <li>• Otherwise, the results are unspecified.</li> </ul>                                                           |
| 35195     | >4 arguments:                | The results are unspecified.                                                                                                                          |
| 35196 XSI |                              | On XSI-conformant systems, combinations of primaries and operators shall be                                                                           |
| 35197     |                              | evaluated using the precedence and associativity rules described previously.                                                                          |
| 35198     |                              | In addition, the string comparison binary primaries ' =' and " != " shall have                                                                        |
| 35199     |                              | a higher precedence than any unary primary.                                                                                                           |
| 35200     | <b>STDIN</b>                 |                                                                                                                                                       |
| 35201     |                              | Not used.                                                                                                                                             |
| 35202     | <b>INPUT FILES</b>           |                                                                                                                                                       |
| 35203     |                              | None.                                                                                                                                                 |
| 35204     | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                       |
| 35205     |                              | The following environment variables shall affect the execution of <i>test</i> :                                                                       |
| 35206     | <i>LANG</i>                  | Provide a default value for the internationalization variables that are unset or null.                                                                |
| 35207     |                              | (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,                                                                                |
| 35208     |                              | Internationalization Variables for the precedence of internationalization variables                                                                   |
| 35209     |                              | used to determine the values of locale categories.)                                                                                                   |
| 35210     | <i>LC_ALL</i>                | If set to a non-empty string value, override the values of all the other                                                                              |
| 35211     |                              | internationalization variables.                                                                                                                       |
| 35212     | <i>LC_CTYPE</i>              | Determine the locale for the interpretation of sequences of bytes of text data as                                                                     |
| 35213     |                              | characters (for example, single-byte as opposed to multi-byte characters in                                                                           |
| 35214     |                              | arguments).                                                                                                                                           |
| 35215     | <i>LC_MESSAGES</i>           |                                                                                                                                                       |
| 35216     |                              | Determine the locale that should be used to affect the format and contents of                                                                         |
| 35217     |                              | diagnostic messages written to standard error.                                                                                                        |
| 35218 XSI | <i>NLSPATH</i>               | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                 |
| 35219     | <b>ASYNCHRONOUS EVENTS</b>   |                                                                                                                                                       |
| 35220     |                              | Default.                                                                                                                                              |

35221 **STDOUT**

35222 Not used.

35223 **STDERR**

35224 The standard error shall be used only for diagnostic messages.

35225 **OUTPUT FILES**

35226 None.

35227 **EXTENDED DESCRIPTION**

35228 None.

35229 **EXIT STATUS**

35230 The following exit values shall be returned:

35231 0 *expression* evaluated to true.35232 1 *expression* evaluated to false or *expression* was missing.

35233 &gt;1 An error occurred.

35234 **CONSEQUENCES OF ERRORS**

35235 Default.

35236 **APPLICATION USAGE**

35237 Scripts should be careful when dealing with user-supplied input that could be confused with  
 35238 primaries and operators. Unless the application writer knows all the cases that produce input to  
 35239 the script, invocations like:

35240 `test "$1" -a "$2"`

35241 should be written as:

35242 `test "$1" && test "$2"`

35243 to avoid problems if a user supplied values such as \$1 set to '!' and \$2 set to the null string.  
 35244 That is, in cases where maximal portability is of concern, replace:

35245 `test expr1 -a expr2`

35246 with:

35247 `test expr1 && test expr2`

35248 and replace:

35249 `test expr1 -o expr2`

35250 with:

35251 `test expr1 || test expr2`

35252 but note that, in *test*, **-a** has higher precedence than **-o** while **"&&"** and **"||"** have equal  
 35253 precedence in the shell.

35254 Parentheses or braces can be used in the shell command language to effect grouping.

35255 Parentheses must be escaped when using *sh*; for example:35256 `test \( expr1 -a expr2 \) -o expr3`

35257 This command is not always portable outside XSI-conformant systems. The following form can  
 35258 be used instead:

```
35259 (test expr1 && test expr2) || test expr3
```

```
35260 The two commands:
```

```
35261 test "$1"
```

```
35262 test ! "$1"
```

```
35263 could not be used reliably on some historical systems. Unexpected results would occur if such a
35264 string expression were used and $1 expanded to '!', '(', or a known unary primary. Better
35265 constructs are:
```

```
35266 test -n "$1"
```

```
35267 test -z "$1"
```

```
35268 respectively.
```

```
35269 Historical systems have also been unreliable given the common construct:
```

```
35270 test "$response" = "expected string"
```

```
35271 One of the following is a more reliable form:
```

```
35272 test "X$response" = "Xexpected string"
```

```
35273 test "expected string" = "$response"
```

```
35274 Note that the second form assumes that expected string could not be confused with any unary
35275 primary. If expected string starts with '-', '(', '!', or even '=', the first form should be used
35276 instead. Using the preceding rules without the XSI marked extensions, any of the three
35277 comparison forms is reliable, given any input. (However, note that the strings are quoted in all
35278 cases.)
```

```
35279 Because the string comparison binary primaries, '=' and '!=', have a higher precedence than
35280 any unary primary in the greater than 4 argument case, unexpected results can occur if
35281 arguments are not properly prepared. For example, in:
```

```
35282 test -d $1 -o -d $2
```

```
35283 If $1 evaluates to a possible directory name of '=', the first three arguments are considered a
35284 string comparison, which shall cause a syntax error when the second -d is encountered. One of
35285 the following forms prevents this; the second is preferred:
```

```
35286 test \(-d "$1" \) -o \(-d "$2" \)
```

```
35287 test -d "$1" || test -d "$2"
```

```
35288 Also in the greater than 4 argument case:
```

```
35289 test "$1" = "bat" -a "$2" = "ball"
```

```
35290 syntax errors occur if $1 evaluates to '(' or '!'. One of the following forms prevents this; the
35291 third is preferred:
```

```
35292 test "X$1" = "Xbat" -a "X$2" = "Xball"
```

```
35293 test "$1" = "bat" && test "$2" = "ball"
```

```
35294 test "X$1" = "Xbat" && test "X$2" = "Xball"
```

## 35295 EXAMPLES

```
35296 1. Exit if there are not two or three arguments (two variations):
```

```
35297 if [$# -ne 2 -a $# -ne 3]; then exit 1; fi
```

```
35298 if [$# -lt 2 -o $# -gt 3]; then exit 1; fi
```

```

35299 2. Perform a mkdir if a directory does not exist:
35300 test ! -d tempdir && mkdir tempdir
35301
35302 3. Wait for a file to become non-readable:
35303 while test -r thefile
35304 do
35305 sleep 30
35306 done
35307 echo '"thefile" is no longer readable'
35308
35309 4. Perform a command if the argument is one of three strings (two variations):
35310 if ["$1" = "pear"] || ["$1" = "grape"] || ["$1" = "apple"]
35311 then
35312 command
35313 fi
35314
35315 case "$1" in
35316 pear|grape|apple) command ;;
35317 esac

```

#### 35315 RATIONALE

35316 The KornShell-derived conditional command (double bracket **[[ ]]**) was removed from the shell  
 35317 command language description in an early proposal. Objections were raised that the real  
 35318 problem is misuse of the *test* command (**I**), and putting it into the shell is the wrong way to fix  
 35319 the problem. Instead, proper documentation and a new shell reserved word (**!**) are sufficient.

35320 Tests that require multiple *test* operations can be done at the shell level using individual  
 35321 invocations of the *test* command and shell logicals, rather than using the error-prone **-o** flag of  
 35322 *test*.

35323 XSI-conformant systems support more than four arguments.

35324 XSI-conformant systems support the combining of primaries with the following constructs:

35325 *expression1* **-a** *expression2*

35326 True if both *expression1* and *expression2* are true.

35327 *expression1* **-o** *expression2*

35328 True if at least one of *expression1* and *expression2* are true.

35329 **(** *expression* **)**

35330 True if *expression* is true.

35331 In evaluating these more complex combined expressions, the following precedence rules are  
 35332 used:

- 35333 • The unary primaries have higher precedence than the algebraic binary primaries.
- 35334 • The unary primaries have lower precedence than the string binary primaries.
- 35335 • The unary and binary primaries have higher precedence than the unary *string* primary.
- 35336 • The **!** operator has higher precedence than the **-a** operator, and the **-a** operator has higher  
 35337 precedence than the **-o** operator.
- 35338 • The **-a** and **-o** operators are left associative.
- 35339 • The parentheses can be used to alter the normal precedence and associativity.

35340 The BSD and System V versions of **-f** are not the same. The BSD definition was:

35341 **-f file** True if *file* exists and is not a directory.

35342 The SVID version (true if the file exists and is a regular file) was chosen for this volume of  
 35343 IEEE Std 1003.1-2001 because its use is consistent with the **-b**, **-c**, **-d**, and **-p** operands (*file* exists  
 35344 and is a specific file type).

35345 The **-e** primary, possessing similar functionality to that provided by the C shell, was added  
 35346 because it provides the only way for a shell script to find out if a file exists without trying to  
 35347 open the file. Since implementations are allowed to add additional file types, a portable script  
 35348 cannot use:

35349 `test -b foo -o -c foo -o -d foo -o -f foo -o -p foo`

35350 to find out if **foo** is an existing file. On historical BSD systems, the existence of a file could be  
 35351 determined by:

35352 `test -f foo -o -d foo`

35353 but there was no easy way to determine that an existing file was a regular file. An early proposal  
 35354 used the KornShell **-a** primary (with the same meaning), but this was changed to **-e** because  
 35355 there were concerns about the high probability of humans confusing the **-a** primary with the **-a**  
 35356 binary operator.

35357 The following options were not included in this volume of IEEE Std 1003.1-2001, although they  
 35358 are provided by some implementations. These operands should not be used by new  
 35359 implementations for other purposes:

35360 **-k file** True if *file* exists and its sticky bit is set.

35361 **-C file** True if *file* is a contiguous file.

35362 **-V file** True if *file* is a version file.

35363 The following option was not included because it was undocumented in most implementations,  
 35364 has been removed from some implementations (including System V), and the functionality is  
 35365 provided by the shell (see Section 2.6.2 (on page 37)).

35366 **-l string** The length of the string *string*.

35367 The **-b**, **-c**, **-g**, **-p**, **-u**, and **-x** operands are derived from the SVID; historical BSD does not  
 35368 provide them. The **-k** operand is derived from System V; historical BSD does not provide it.

35369 On historical BSD systems, `test -w directory` always returned false because `test` tried to open the  
 35370 directory for writing, which always fails.

35371 Some additional primaries newly invented or from the KornShell appeared in an early proposal  
 35372 as part of the conditional command (**[I I]**): `s1 > s2`, `s1 < s2`, `str = pattern`, `str != pattern`, `f1 -nt f2`, `f1`  
 35373 `-ot f2`, and `f1 -ef f2`. They were not carried forward into the `test` utility when the conditional  
 35374 command was removed from the shell because they have not been included in the `test` utility  
 35375 built into historical implementations of the `sh` utility.

35376 The **-t file\_descriptor** primary is shown with a mandatory argument because the grammar is  
 35377 ambiguous if it can be omitted. Historical implementations have allowed it to be omitted,  
 35378 providing a default of 1.

35379 **FUTURE DIRECTIONS**

35380 None.

35381 **SEE ALSO**35382           Section 1.7.1.4 (on page 4), *find*35383 **CHANGE HISTORY**

35384           First released in Issue 2.

35385 **Issue 5**

35386           The FUTURE DIRECTIONS section is added.

35387 **Issue 6**35388           The **-h** operand is added for symbolic links, and access permission requirements are clarified for  
35389           the **-r**, **-w**, and **-x** operands to align with the IEEE P1003.2b draft standard.

35390           The normative text is reworded to avoid use of the term “must” for application requirements.

35391           The **-L** and **-S** operands are added for symbolic links and sockets.35392           IEEE Std 1003.1-2001 / Cor 1-2002, item XCU / TC1 / D6 / 38 is applied, adding XSI margin marking |  
35393           and shading to a line in the OPERANDS section referring to the use of parentheses as arguments |  
35394           to the *test* utility. |

35395 **NAME**

35396           time — time a simple command

35397 **SYNOPSIS**35398 UP       time [-p] *utility* [*argument...*]

35399

35400 **DESCRIPTION**

35401       The *time* utility shall invoke the utility named by the *utility* operand with arguments supplied as  
 35402       the *argument* operands and write a message to standard error that lists timing statistics for the  
 35403       utility. The message shall include the following information:

- 35404           • The elapsed (real) time between invocation of *utility* and its termination.
- 35405           • The User CPU time, equivalent to the sum of the *tms\_utime* and *tms\_cutime* fields returned by  
 35406           the *times()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 for the  
 35407           process in which *utility* is executed.
- 35408           • The System CPU time, equivalent to the sum of the *tms\_stime* and *tms\_cstime* fields returned  
 35409           by the *times()* function for the process in which *utility* is executed.

35410       The precision of the timing shall be no less than the granularity defined for the size of the clock  
 35411       tick unit on the system, but the results shall be reported in terms of standard time units (for  
 35412       example, 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

35413       When *time* is used as part of a pipeline, the times reported are unspecified, except when it is the  
 35414       sole command within a grouping command (see Section 2.9.4.1 (on page 52)) in that pipeline. For  
 35415       example, the commands on the left are unspecified; those on the right report on utilities **a** and **c**,  
 35416       respectively:

```
35417 time a | b | c { time a } | b | c
35418 a | b | time c a | b | (time c)
```

35419 **OPTIONS**

35420       The *time* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 35421       12.2, Utility Syntax Guidelines.

35422       The following option shall be supported:

35423       **-p**           Write the timing output to standard error in the format shown in the STDERR  
 35424       section.

35425 **OPERANDS**

35426       The following operands shall be supported:

35427       *utility*       The name of a utility that is to be invoked. If the *utility* operand names any of the  
 35428       special built-in utilities in Section 2.14 (on page 64), the results are undefined.

35429       *argument*     Any string to be supplied as an argument when invoking the utility named by the  
 35430       *utility* operand.

35431 **STDIN**

35432       Not used.

35433 **INPUT FILES**

35434       None.

35435 **ENVIRONMENT VARIABLES**

35436 The following environment variables shall affect the execution of *time*:

35437 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 35438 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 35439 Internationalization Variables for the precedence of internationalization variables  
 35440 used to determine the values of locale categories.)

35441 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 35442 internationalization variables.

35443 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 35444 characters (for example, single-byte as opposed to multi-byte characters in  
 35445 arguments).

35446 **LC\_MESSAGES**  
 35447 Determine the locale that should be used to affect the format and contents of  
 35448 diagnostic and informative messages written to standard error.

35449 **LC\_NUMERIC**  
 35450 Determine the locale for numeric formatting.

35451 **NSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

35452 **PATH** Determine the search path that shall be used to locate the utility to be invoked; see  
 35453 the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment  
 35454 Variables.

35455 **ASYNCHRONOUS EVENTS**

35456 Default.

35457 **STDOUT**

35458 Not used.

35459 **STDERR**

35460 The standard error shall be used to write the timing statistics. If **-p** is specified, the following  
 35461 format shall be used in the POSIX locale:

35462 "real %f\nuser %f\nsys %f\n", <real seconds>, <user seconds>,  
 35463 <system seconds>

35464 where each floating-point number shall be expressed in seconds. The precision used may be less  
 35465 than the default six digits of %f, but shall be sufficiently precise to accommodate the size of the  
 35466 clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits  
 35467 shall follow the radix character). The number of digits following the radix character shall be no  
 35468 less than one, even if this always results in a trailing zero. The implementation may append  
 35469 white space and additional information following the format shown here.

35470 **OUTPUT FILES**

35471 None.

35472 **EXTENDED DESCRIPTION**

35473 None.

35474 **EXIT STATUS**

35475 If the *utility* utility is invoked, the exit status of *time* shall be the exit status of *utility*; otherwise,  
 35476 the *time* utility shall exit with one of the following values:

35477 1-125 An error occurred in the *time* utility.



35478 126 The utility specified by *utility* was found but could not be invoked.

35479 127 The utility specified by *utility* could not be found.

## 35480 CONSEQUENCES OF ERRORS

35481 Default.

## 35482 APPLICATION USAGE

35483 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
 35484 an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
 35485 utility exited with an error indication”. The value 127 was chosen because it is not commonly  
 35486 used for other meanings; most utilities use small values for “normal error conditions” and the  
 35487 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
 35488 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
 35489 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
 35490 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
 35491 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
 35492 any other reason.

## 35493 EXAMPLES

35494 It is frequently desirable to apply *time* to pipelines or lists of commands. This can be done by  
 35495 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and  
 35496 the *time* applies to everything in the file.

35497 Alternatively, the following command can be used to apply *time* to a complex command:

35498 `time sh -c 'complex-command-line'`

## 35499 RATIONALE

35500 When the *time* utility was originally proposed to be included in the ISO POSIX-2: 1993 standard,  
 35501 questions were raised about its suitability for inclusion on the grounds that it was not useful for  
 35502 conforming applications, specifically:

- 35503 • The underlying CPU definitions from the System Interfaces volume of IEEE Std 1003.1-2001  
 35504 are vague, so the numeric output could not be compared accurately between systems or even  
 35505 between invocations.
- 35506 • The creation of portable benchmark programs was outside the scope this volume of  
 35507 IEEE Std 1003.1-2001.

35508 However, *time* does fit in the scope of user portability. Human judgement can be applied to the  
 35509 analysis of the output, and it could be very useful in hands-on debugging of applications or in  
 35510 providing subjective measures of system performance. Hence it has been included in this  
 35511 volume of IEEE Std 1003.1-2001.

35512 The default output format has been left unspecified because historical implementations differ  
 35513 greatly in their style of depicting this numeric output. The **-p** option was invented to provide  
 35514 scripts with a common means of obtaining this information.

35515 In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather  
 35516 than just a simple command. The POSIX definition has been worded to allow this  
 35517 implementation. Consideration was given to invalidating this approach because of the historical  
 35518 model from the C shell and System V shell. However, since the System V *time* utility historically  
 35519 has not produced accurate results in pipeline timing (because the constituent processes are not  
 35520 all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to  
 35521 break historical KornShell usage.

35522 The term *utility* is used, rather than *command*, to highlight the fact that shell compound  
 35523 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*

35524 includes user application programs and shell scripts, not just the standard utilities.

35525 **FUTURE DIRECTIONS**

35526 None.

35527 **SEE ALSO**

35528 Chapter 2 (on page 29), *sh*, the System Interfaces volume of IEEE Std 1003.1-2001, *times()*

35529 **CHANGE HISTORY**

35530 First released in Issue 2.

35531 **Issue 6**

35532 This utility is marked as part of the User Portability Utilities option.

35533 **NAME**

35534 touch — change file access and modification times

35535 **SYNOPSIS**35536 touch [-acm] [ -r *ref\_file* | -t *time*] *file*...35537 **DESCRIPTION**

35538 The *touch* utility shall change the modification times, access times, or both of files. The  
 35539 modification time shall be equivalent to the value of the *st\_mtime* member of the **stat** structure  
 35540 for a file, as described in the System Interfaces volume of IEEE Std 1003.1-2001; the access time  
 35541 shall be equivalent to the value of *st\_atime*.

35542 The time used can be specified by the **-t** *time* option-argument, the corresponding time fields of  
 35543 the file referenced by the **-r** *ref\_file* option-argument, or the *date\_time* operand, as specified in the  
 35544 following sections. If none of these are specified, *touch* shall use the current time (the value  
 35545 returned by the equivalent of the *time()* function defined in the System Interfaces volume of  
 35546 IEEE Std 1003.1-2001).

35547 For each *file* operand, *touch* shall perform actions equivalent to the following functions defined  
 35548 in the System Interfaces volume of IEEE Std 1003.1-2001:

- 35549 1. If *file* does not exist, a *creat()* function call is made with the *file* operand used as the *path*  
 35550 argument and the value of the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, S\_IRGRP,  
 35551 S\_IWGRP, S\_IROTH, and S\_IWOTH used as the *mode* argument.
- 35552 2. The *utime()* function is called with the following arguments:
  - 35553 a. The *file* operand is used as the *path* argument.
  - 35554 b. The **utimbuf** structure members *actime* and *modtime* are determined as described in  
 35555 the OPTIONS section.

35556 **OPTIONS**

35557 The *touch* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 35558 12.2, Utility Syntax Guidelines.

35559 The following options shall be supported:

- |       |                           |                                                                                                 |
|-------|---------------------------|-------------------------------------------------------------------------------------------------|
| 35560 | <b>-a</b>                 | Change the access time of <i>file</i> . Do not change the modification time unless <b>-m</b> is |
| 35561 |                           | also specified.                                                                                 |
| 35562 | <b>-c</b>                 | Do not create a specified <i>file</i> if it does not exist. Do not write any diagnostic         |
| 35563 |                           | messages concerning this condition.                                                             |
| 35564 | <b>-m</b>                 | Change the modification time of <i>file</i> . Do not change the access time unless <b>-a</b> is |
| 35565 |                           | also specified.                                                                                 |
| 35566 | <b>-r</b> <i>ref_file</i> | Use the corresponding time of the file named by the pathname <i>ref_file</i> instead of         |
| 35567 |                           | the current time.                                                                               |
| 35568 | <b>-t</b> <i>time</i>     | Use the specified <i>time</i> instead of the current time. The option-argument shall be a       |
| 35569 |                           | decimal number of the form:                                                                     |

35570 [ [CC] YY] MMDDhhmm [ . SS]

35571 where each two digits represents the following:

- |       |           |                                |
|-------|-----------|--------------------------------|
| 35572 | <i>MM</i> | The month of the year [01,12]. |
| 35573 | <i>DD</i> | The day of the month [01,31].  |

35574            *hh*        The hour of the day [00,23].

35575            *mm*        The minute of the hour [00,59].

35576            *CC*        The first two digits of the year (the century).

35577            *YY*        The second two digits of the year.

35578            *SS*        The second of the minute [00,60].

35579            Both *CC* and *YY* shall be optional. If neither is given, the current year shall be  
 35580            assumed. If *YY* is specified, but *CC* is not, *CC* shall be derived as follows:

|       |                         |                           |
|-------|-------------------------|---------------------------|
| 35581 | <b>If <i>YY</i> is:</b> | <b><i>CC</i> becomes:</b> |
| 35582 | [69,99]                 | 19                        |
| 35583 | [00,68]                 | 20                        |

35584            **Note:**        It is expected that in a future version of IEEE Std 1003.1-2001 the default  
 35585            century inferred from a 2-digit year will change. (This would apply to all  
 35586            commands accepting a 2-digit year as input.)

35587            The resulting time shall be affected by the value of the *TZ* environment variable. If  
 35588            the resulting time value precedes the Epoch, *touch* shall exit immediately with an  
 35589            error status. The range of valid times past the Epoch is implementation-defined,  
 35590            but it shall extend to at least the time 0 hours, 0 minutes, 0 seconds, January 1,  
 35591            2038, Coordinated Universal Time. Some implementations may not be able to  
 35592            represent dates beyond January 18, 2038, because they use **signed int** as a time  
 35593            holder.

35594            The range for *SS* is [00,60] rather than [00,59] because of leap seconds. If *SS* is 60,  
 35595            and the resulting time, as affected by the *TZ* environment variable, does not refer  
 35596            to a leap second, the resulting time shall be one second after a time where *SS* is 59.  
 35597            If *SS* is not given a value, it is assumed to be zero.

35598            If neither the **-a** nor **-m** options were specified, *touch* shall behave as if both the **-a** and **-m**  
 35599            options were specified.

## 35600 OPERANDS

35601            The following operands shall be supported:

35602            *file*            A pathname of a file whose times shall be modified.

## 35603 STDIN

35604            Not used.

## 35605 INPUT FILES

35606            None.

## 35607 ENVIRONMENT VARIABLES

35608            The following environment variables shall affect the execution of *touch*:

35609            *LANG*            Provide a default value for the internationalization variables that are unset or null.  
 35610            (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 35611            Internationalization Variables for the precedence of internationalization variables  
 35612            used to determine the values of locale categories.)

35613            *LC\_ALL*          If set to a non-empty string value, override the values of all the other  
 35614            internationalization variables.

35615            *LC\_CTYPE*        Determine the locale for the interpretation of sequences of bytes of text data as  
 35616            characters (for example, single-byte as opposed to multi-byte characters in

arguments).

**LC\_MESSAGES**  
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

**NLSPATH** XSI Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

**TZ**  
Determine the timezone to be used for interpreting the *time* option-argument. If *TZ* is unset or null, an unspecified default timezone shall be used.

**ASYNCHRONOUS EVENTS**  
Default.

**STDOUT**  
Not used.

**STDERR**  
The standard error shall be used only for diagnostic messages.

**OUTPUT FILES**  
None.

**EXTENDED DESCRIPTION**  
None.

**EXIT STATUS**  
The following exit values shall be returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

**CONSEQUENCES OF ERRORS**  
Default.

**APPLICATION USAGE**  
The interpretation of time is taken to be *seconds since the Epoch* (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.14, Seconds Since the Epoch). It should be noted that implementations conforming to the System Interfaces volume of IEEE Std 1003.1-2001 do not take leap seconds into account when computing seconds since the Epoch. When *SS=60* is used, the resulting time always refers to 1 plus *seconds since the Epoch* for a time when *SS=59*.

Although the **-t time** option-argument specifies values in 1969, the access time and modification time fields are defined in terms of seconds since the Epoch (00:00:00 on 1 January 1970 UTC). Therefore, depending on the value of *TZ* when *touch* is run, there is never more than a few valid hours in 1969 and there need not be any valid times in 1969.

One ambiguous situation occurs if **-t time** is not specified, **-r ref\_file** is not specified, and the first operand is an eight or ten-digit decimal number. A portable script can avoid this problem by using:

```
touch -- file
```

or:

```
touch ./file
```

in this case.

35657 **EXAMPLES**

35658           None.

35659 **RATIONALE**

35660           The functionality of *touch* is described almost entirely through references to functions in the  
 35661           System Interfaces volume of IEEE Std 1003.1-2001. In this way, there is no duplication of effort  
 35662           required for describing such side effects as the relationship of user IDs to the user database,  
 35663           permissions, and so on.

35664           There are some significant differences between the *touch* utility in this volume of  
 35665           IEEE Std 1003.1-2001 and those in System V and BSD systems. They are upwards-compatible for  
 35666           historical applications from both implementations:

35667           1. In System V, an ambiguity exists when a pathname that is a decimal number leads the  
 35668           operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be  
 35669           *touched* to the current time. The **-t** *time* construct solves these problems for future  
 35670           conforming applications (note that the **-t** option is not historical practice).

35671           2. The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is  
 35672           treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates following  
 35673           the Epoch was included as recognition that some implementations are not able to  
 35674           represent dates beyond 18 January 2038 because they use **signed int** as a time holder.

35675           The **-r** option was added because several comments requested this capability. This option was  
 35676           named **-f** in an early proposal, but was changed because the **-f** option is used in the BSD version  
 35677           of *touch* with a different meaning.

35678           At least one historical implementation of *touch* incremented the exit code if **-c** was specified and  
 35679           the file did not exist. This volume of IEEE Std 1003.1-2001 requires exit status zero if no errors  
 35680           occur.

35681 **FUTURE DIRECTIONS**35682           Applications should use the **-r** or **-t** options.35683 **SEE ALSO**

35684           *date*, the System Interfaces volume of IEEE Std 1003.1-2001, *creat()*, *time()*, *utime()*, the Base  
 35685           Definitions volume of IEEE Std 1003.1-2001, **<sys/stat.h>**

35686 **CHANGE HISTORY**

35687           First released in Issue 2.

35688 **Issue 6**35689           The obsolescent *date\_time* operand is removed.

35690           The Open Group Corrigendum U027/1 is applied. This extends the range of valid time past the  
 35691           Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal  
 35692           Time. This is a new requirement on POSIX implementations.

35693           The range for seconds is changed from [00,61] to [00,60] to align with the ISO/IEC 9899:1999  
 35694           standard, and to allow for positive leap seconds.

35695 **NAME**

35696           tput — change terminal characteristics

35697 **SYNOPSIS**35698 UP       tput [-T *type*] *operand...*

35699

35700 **DESCRIPTION**

35701       The *tput* utility shall display terminal-dependent information. The manner in which this  
 35702       information is retrieved is unspecified. The information displayed shall clear the terminal screen,  
 35703       initialize the user's terminal, or reset the user's terminal, depending on the operand given. The  
 35704       exact consequences of displaying this information are unspecified.

35705 **OPTIONS**

35706       The *tput* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 35707       12.2, Utility Syntax Guidelines.

35708       The following option shall be supported:

35709       **-T *type***     Indicate the type of terminal. If this option is not supplied and the *TERM* variable  
 35710       is unset or null, an unspecified default terminal type shall be used. The setting of  
 35711       *type* shall take precedence over the value in *TERM*.

35712 **OPERANDS**

35713       The following strings shall be supported as operands by the implementation in the POSIX locale:

35714       **clear**       Display the clear-screen sequence.

35715       **init**        Display the sequence that initializes the user's terminal in an implementation-  
 35716       defined manner.

35717       **reset**       Display the sequence that resets the user's terminal in an implementation-defined  
 35718       manner.

35719       If a terminal does not support any of the operations described by these operands, this shall not  
 35720       be considered an error condition.

35721 **STDIN**

35722       Not used.

35723 **INPUT FILES**

35724       None.

35725 **ENVIRONMENT VARIABLES**

35726       The following environment variables shall affect the execution of *tput*:

35727       **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 35728       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 35729       Internationalization Variables for the precedence of internationalization variables  
 35730       used to determine the values of locale categories.)

35731       **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 35732       internationalization variables.

35733       **LC\_CTYPE**    Determine the locale for the interpretation of sequences of bytes of text data as  
 35734       characters (for example, single-byte as opposed to multi-byte characters in  
 35735       arguments).

35736       **LC\_MESSAGES**

35737       Determine the locale that should be used to affect the format and contents of  
 35738       diagnostic messages written to standard error.

35739 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

35740 **TERM** Determine the terminal type. If this variable is unset or null, and if the **-T** option is  
35741 not specified, an unspecified default terminal type shall be used.

35742 **ASYNCHRONOUS EVENTS**

35743 Default.

35744 **STDOUT**

35745 If standard output is a terminal device, it may be used for writing the appropriate sequence to  
35746 clear the screen or reset or initialize the terminal. If standard output is not a terminal device,  
35747 undefined results occur.

35748 **STDERR**

35749 The standard error shall be used only for diagnostic messages.

35750 **OUTPUT FILES**

35751 None.

35752 **EXTENDED DESCRIPTION**

35753 None.

35754 **EXIT STATUS**

35755 The following exit values shall be returned:

35756 0 The requested string was written successfully.

35757 1 Unspecified.

35758 2 Usage error.

35759 3 No information is available about the specified terminal type.

35760 4 The specified operand is invalid.

35761 >4 An error occurred.

35762 **CONSEQUENCES OF ERRORS**

35763 If one of the operands is not available for the terminal, *tput* continues processing the remaining  
35764 operands.

35765 **APPLICATION USAGE**

35766 The difference between resetting and initializing a terminal is left unspecified, as they vary  
35767 greatly based on hardware types. In general, resetting is a more severe action.

35768 Some terminals use control characters to perform the stated functions, and on such terminals it  
35769 might make sense to use *tput* to store the initialization strings in a file or environment variable  
35770 for later use. However, because other terminals might rely on system calls to do this work, the  
35771 standard output cannot be used in a portable manner, such as the following non-portable  
35772 constructs:

35773 ClearVar='tput clear'  
35774 tput reset | mailx -s "Wake Up" ddg

35775 **EXAMPLES**

35776 1. Initialize the terminal according to the type of terminal in the environmental variable  
35777 *TERM*. This command can be included in a **.profile** file.

35778 tput init

35779 2. Reset a 450 terminal.



35780                   tput -T 450 reset

35781 **RATIONALE**

35782                   The list of operands was reduced to a minimum for the following reasons:

- 35783                   • The only features chosen were those that were likely to be used by human users interacting  
35784                   with a terminal.
- 35785                   • Specifying the full *terminfo* set was not considered desirable, but the standard developers did  
35786                   not want to select among operands.
- 35787                   • This volume of IEEE Std 1003.1-2001 does not attempt to provide applications with  
35788                   sophisticated terminal handling capabilities, as that falls outside of its assigned scope and  
35789                   intersects with the responsibilities of other standards bodies.

35790                   The difference between resetting and initializing a terminal is left unspecified as this varies  
35791                   greatly based on hardware types. In general, resetting is a more severe action.

35792                   The exit status of 1 is historically reserved for finding out if a Boolean operand is not set.  
35793                   Although the operands were reduced to a minimum, the exit status of 1 should still be reserved  
35794                   for the Boolean operands, for those sites that wish to support them.

35795 **FUTURE DIRECTIONS**

35796                   None.

35797 **SEE ALSO**

35798                   *stty*, *tabs*

35799 **CHANGE HISTORY**

35800                   First released in Issue 4.

35801 **Issue 6**

35802                   This utility is marked as part of the User Portability Utilities option.

35803 **NAME**

35804 tr — translate characters

35805 **SYNOPSIS**35806 tr [-c | -C] [-s] *string1 string2*35807 tr -s [-c | -C] *string1*35808 tr -d [-c | -C] *string1*35809 tr -ds [-c | -C] *string1 string2*35810 **DESCRIPTION**

35811 The *tr* utility shall copy the standard input to the standard output with substitution or deletion  
 35812 of selected characters. The options specified and the *string1* and *string2* operands shall control  
 35813 translations that occur while copying characters and single-character collating elements.

35814 **OPTIONS**

35815 The *tr* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 35816 Utility Syntax Guidelines.

35817 The following options shall be supported:

35818 **-c** Complement the set of values specified by *string1*. See the EXTENDED  
 35819 DESCRIPTION section.

35820 **-C** Complement the set of characters specified by *string1*. See the EXTENDED  
 35821 DESCRIPTION section.

35822 **-d** Delete all occurrences of input characters that are specified by *string1*.

35823 **-s** Replace instances of repeated characters with a single character, as described in the  
 35824 EXTENDED DESCRIPTION section.

35825 **OPERANDS**

35826 The following operands shall be supported:

35827 *string1, string2*

35828 Translation control strings. Each string shall represent a set of characters to be  
 35829 converted into an array of characters used for the translation. For a detailed  
 35830 description of how the strings are interpreted, see the EXTENDED DESCRIPTION  
 35831 section.

35832 **STDIN**

35833 The standard input can be any type of file.

35834 **INPUT FILES**

35835 None.

35836 **ENVIRONMENT VARIABLES**35837 The following environment variables shall affect the execution of *tr*:

35838 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 35839 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 35840 Internationalization Variables for the precedence of internationalization variables  
 35841 used to determine the values of locale categories.)

35842 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 35843 internationalization variables.

35844 **LC\_COLLATE**

35845 Determine the locale for the behavior of range expressions and equivalence classes.

|       |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 35846 | <i>LC_CTYPE</i>             | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and the behavior of character classes.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35847 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35848 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35849 | <i>LC_MESSAGES</i>          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35850 |                             | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 35851 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35852 | XSI <i>NLSPATH</i>          | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 35853 | <b>ASYNCHRONOUS EVENTS</b>  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35854 |                             | Default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 35855 | <b>STDOUT</b>               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35856 |                             | The <i>tr</i> output shall be identical to the input, with the exception of the specified transformations.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35857 | <b>STDERR</b>               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35858 |                             | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 35859 | <b>OUTPUT FILES</b>         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35860 |                             | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 35861 | <b>EXTENDED DESCRIPTION</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35862 |                             | The operands <i>string1</i> and <i>string2</i> (if specified) define two arrays of characters. The constructs in the following list can be used to specify characters or single-character collating elements. If any of the constructs result in multi-character collating elements, <i>tr</i> shall exclude, without a diagnostic, those multi-character elements from the resulting array.                                                                                                                                                                                                                                                            |
| 35863 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35864 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35865 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35866 | <i>character</i>            | Any character not described by one of the conventions below shall represent itself.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 35867 | <i>\octal</i>               | Octal sequences can be used to represent characters with specific coded values. An octal sequence shall consist of a backslash followed by the longest sequence of one, two, or three-octal-digit characters (01234567). The sequence shall cause the value whose encoding is represented by the one, two, or three-digit octal integer to be placed into the array. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading ' <i>\</i> ' for each byte. |
| 35868 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35869 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35870 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35871 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35872 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35873 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35874 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35875 | <i>\character</i>           | The backslash-escape sequences in the Base Definitions volume of IEEE Std 1003.1-2001, Table 5-1, Escape Sequences and Associated Actions (' <i>\</i> <i>\</i> ', ' <i>\a</i> ', ' <i>\b</i> ', ' <i>\f</i> ', ' <i>\n</i> ', ' <i>\r</i> ', ' <i>\t</i> ', ' <i>\v</i> ') shall be supported. The results of using any other character, other than an octal digit, following the backslash are unspecified.                                                                                                                                                                                                                                            |
| 35876 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35877 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35878 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35879 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35880 | <i>c-c</i>                  | In the POSIX locale, this construct shall represent the range of collating elements between the range endpoints (as long as neither endpoint is an octal sequence of the form <i>\octal</i> ), inclusive, as defined by the collation sequence. The characters or collating elements in the range shall be placed in the array in ascending collation sequence. If the second endpoint precedes the starting endpoint in the collation sequence, it is unspecified whether the range of collating elements is empty, or this construct is treated as invalid. In locales other than the POSIX locale, this construct has unspecified behavior.          |
| 35881 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35882 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35883 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35884 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35885 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35886 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35887 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35888 |                             | If either or both of the range endpoints are octal sequences of the form <i>\octal</i> , this shall represent the range of specific coded values between the two range endpoints, inclusive.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 35889 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 35890 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

|       |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 35891 | <code>[:class:]</code> | Represents all characters belonging to the defined character class, as defined by the current setting of the <code>LC_CTYPE</code> locale category. The following character class names shall be accepted when specified in <i>string1</i> :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 35892 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35893 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35894 |                        | <b>alnum</b> <b>blank</b> <b>digit</b> <b>lower</b> <b>punct</b> <b>upper</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 35895 |                        | <b>alpha</b> <b>cntrl</b> <b>graph</b> <b>print</b> <b>space</b> <b>xdigit</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 35896 | XS1                    | In addition, character class expressions of the form <code>[:name:]</code> shall be recognized in those locales where the <i>name</i> keyword has been given a <b>charclass</b> definition in the <code>LC_CTYPE</code> category.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 35897 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35898 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35899 |                        | When both the <code>-d</code> and <code>-s</code> options are specified, any of the character class names shall be accepted in <i>string2</i> . Otherwise, only character class names <b>lower</b> or <b>upper</b> are valid in <i>string2</i> and then only if the corresponding character class ( <b>upper</b> and <b>lower</b> , respectively) is specified in the same relative position in <i>string1</i> . Such a specification shall be interpreted as a request for case conversion. When <code>[:lower:]</code> appears in <i>string1</i> and <code>[:upper:]</code> appears in <i>string2</i> , the arrays shall contain the characters from the <b>toupper</b> mapping in the <code>LC_CTYPE</code> category of the current locale. When <code>[:upper:]</code> appears in <i>string1</i> and <code>[:lower:]</code> appears in <i>string2</i> , the arrays shall contain the characters from the <b>tolower</b> mapping in the <code>LC_CTYPE</code> category of the current locale. The first character from each mapping pair shall be in the array for <i>string1</i> and the second character from each mapping pair shall be in the array for <i>string2</i> in the same relative position. |
| 35900 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35901 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35902 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35903 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35904 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35905 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35906 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35907 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35908 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35909 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35910 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35911 |                        | Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 35912 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35913 |                        | If the name specified for <i>class</i> does not define a valid character class in the current locale, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 35914 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35915 | <code>[=equiv=]</code> | Represents all characters or collating elements belonging to the same equivalence class as <i>equiv</i> , as defined by the current setting of the <code>LC_COLLATE</code> locale category. An equivalence class expression shall be allowed only in <i>string1</i> , or in <i>string2</i> when it is being used by the combined <code>-d</code> and <code>-s</code> options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 35916 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35917 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35918 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35919 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35920 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35921 | <code>[x*n]</code>     | Represents <i>n</i> repeated occurrences of the character <i>x</i> . Because this expression is used to map multiple characters to one, it is only valid when it occurs in <i>string2</i> . If <i>n</i> is omitted or is zero, it shall be interpreted as large enough to extend the <i>string2</i> -based sequence to the length of the <i>string1</i> -based sequence. If <i>n</i> has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 35922 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35923 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35924 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35925 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35926 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35927 |                        | When the <code>-d</code> option is not specified:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 35928 |                        | • Each input character found in the array specified by <i>string1</i> shall be replaced by the character in the same relative position in the array specified by <i>string2</i> . When the array specified by <i>string2</i> is shorter than the one specified by <i>string1</i> , the results are unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35929 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35930 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35931 |                        | • If the <code>-C</code> option is specified, the complements of the characters specified by <i>string1</i> (the set of all characters in the current character set, as defined by the current setting of <code>LC_CTYPE</code> , except for those actually specified in the <i>string1</i> operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of <code>LC_COLLATE</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 35932 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35933 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35934 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 35935 |                        | • If the <code>-c</code> option is specified, the complement of the values specified by <i>string1</i> shall be placed in the array in ascending order by binary value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 35936 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

35937 • Because the order in which characters specified by character class expressions or equivalence  
 35938 class expressions is undefined, such expressions should only be used if the intent is to map  
 35939 several characters into one. An exception is case conversion, as described previously.

35940 When the **-d** option is specified:

- 35941 • Input characters found in the array specified by *string1* shall be deleted.
- 35942 • When the **-C** option is specified with **-d**, all characters except those specified by *string1* shall  
 35943 be deleted. The contents of *string2* are ignored, unless the **-s** option is also specified.
- 35944 • When the **-c** option is specified with **-d**, all values except those specified by *string1* shall be  
 35945 deleted. The contents of *string2* shall be ignored, unless the **-s** option is also specified.
- 35946 • The same string cannot be used for both the **-d** and the **-s** option; when both options are  
 35947 specified, both *string1* (used for deletion) and *string2* (used for squeezing) shall be required.

35948 When the **-s** option is specified, after any deletions or translations have taken place, repeated  
 35949 sequences of the same character shall be replaced by one occurrence of the same character, if the  
 35950 character is found in the array specified by the last operand. If the last operand contains a  
 35951 character class, such as the following example:

35952 `tr -s '[:space:]'`

35953 the last operand's array shall contain all of the characters in that character class. However, in a  
 35954 case conversion, as described previously, such as:

35955 `tr -s '[:upper:]' '[:lower:]'`

35956 the last operand's array shall contain only those characters defined as the second characters in  
 35957 each of the **toupper** or **tolower** character pairs, as appropriate.

35958 An empty string used for *string1* or *string2* produces undefined results.

#### 35959 EXIT STATUS

35960 The following exit values shall be returned:

35961 0 All input was processed successfully.

35962 >0 An error occurred.

#### 35963 CONSEQUENCES OF ERRORS

35964 Default.

#### 35965 APPLICATION USAGE

35966 If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the shell.

35967 If an ordinary digit (representing itself) is to follow an octal sequence, the octal sequence must  
 35968 use the full three digits to avoid ambiguity.

35969 When *string2* is shorter than *string1*, a difference results between historical System V and BSD  
 35970 systems. A BSD system pads *string2* with the last character found in *string2*. Thus, it is possible  
 35971 to do the following:

35972 `tr 0123456789 d`

35973 which would translate all digits to the letter 'd'. Since this area is specifically unspecified in  
 35974 this volume of IEEE Std 1003.1-2001, both the BSD and System V behaviors are allowed, but a  
 35975 conforming application cannot rely on the BSD behavior. It would have to code the example in  
 35976 the following way:

35977 `tr 0123456789 '[d*]'`

35978 It should be noted that, despite similarities in appearance, the string operands used by *tr* are not  
35979 regular expressions.

35980 Unlike some historical implementations, this definition of the *tr* utility correctly processes NUL  
35981 characters in its input stream. NUL characters can be stripped by using:

35982 `tr -d '\000'`

### 35983 EXAMPLES

35984 1. The following example creates a list of all words in **file1** one per line in **file2**, where a word  
35985 is taken to be a maximal string of letters.

35986 `tr -cs "[:alpha:]" "[\n*]" <file1 >file2`

35987 2. The next example translates all lowercase characters in **file1** to uppercase and writes the  
35988 results to standard output.

35989 `tr "[:lower:]" "[:upper:]" <file1`

35990 3. This example uses an equivalence class to identify accented variants of the base character  
35991 'e' in **file1**, which are stripped of diacritical marks and written to **file2**.

35992 `tr "[=e=]" e <file1 >file2`

### 35993 RATIONALE

35994 In some early proposals, an explicit option **-n** was added to disable the historical behavior of  
35995 stripping NUL characters from the input. It was considered that automatically stripping NUL  
35996 characters from the input was not correct functionality. However, the removal of **-n** in a later  
35997 proposal does not remove the requirement that *tr* correctly process NUL characters in its input  
35998 stream. NUL characters can be stripped by using `tr -d '\000'`.

35999 Historical implementations of *tr* differ widely in syntax and behavior. For example, the BSD  
36000 version has not needed the bracket characters for the repetition sequence. The *tr* utility syntax is  
36001 based more closely on the System V and XPG3 model while attempting to accommodate  
36002 historical BSD implementations. In the case of the short *string2* padding, the decision was to  
36003 unspecify the behavior and preserve System V and XPG3 scripts, which might find difficulty  
36004 with the BSD method. The assumption was made that BSD users of *tr* have to make  
36005 accommodations to meet the syntax defined here. Since it is possible to use the repetition  
36006 sequence to duplicate the desired behavior, whereas there is no simple way to achieve the  
36007 System V method, this was the correct, if not desirable, approach.

36008 The use of octal values to specify control characters, while having historical precedents, is not  
36009 portable. The introduction of escape sequences for control characters should provide the  
36010 necessary portability. It is recognized that this may cause some historical scripts to break.

36011 An early proposal included support for multi-character collating elements. It was pointed out  
36012 that, while *tr* does employ some syntactical elements from REs, the aim of *tr* is quite different;  
36013 ranges, for example, do not have a similar meaning ("any of the chars in the range matches",  
36014 *versus* "translate each character in the range to the output counterpart"). As a result, the  
36015 previously included support for multi-character collating elements has been removed. What  
36016 remains are ranges in current collation order (to support, for example, accented characters),  
36017 character classes, and equivalence classes.

36018 In XPG3 the `[:class:]` and `[equiv=]` conventions are shown with double brackets, as in RE syntax.  
36019 However, *tr* does not implement RE principles; it just borrows part of the syntax. Consequently,  
36020 `[:class:]` and `[equiv=]` should be regarded as syntactical elements on a par with `[x*n]`, which is  
36021 not an RE bracket expression.

36022 The standard developers will consider changes to *tr* that allow it to translate characters between  
 36023 different character encodings, or they will consider providing a new utility to accomplish this.

36024 On historical System V systems, a range expression requires enclosing square-brackets, such as:

36025 `tr '[a-z]' '[A-Z]'`

36026 However, BSD-based systems did not require the brackets, and this convention is used here to  
 36027 avoid breaking large numbers of BSD scripts:

36028 `tr a-z A-Z`

36029 The preceding System V script will continue to work because the brackets, treated as regular  
 36030 characters, are translated to themselves. However, any System V script that relied on "a-z"  
 36031 representing the three characters 'a', '-', and 'z' have to be rewritten as "az-".

36032 The ISO POSIX-2: 1993 standard had a `-c` option that behaved similarly to the `-C` option, but did  
 36033 not supply functionality equivalent to the `-c` option specified in IEEE Std 1003.1-2001. This  
 36034 meant that historical practice of being able to specify `tr -d\200-\377` (which would delete all  
 36035 bytes with the top bit set) would have no effect because, in the C locale, bytes with the values  
 36036 octal 200 to octal 377 are not characters.

36037 The earlier version also said that octal sequences referred to collating elements and could be  
 36038 placed adjacent to each other to specify multi-byte characters. However, it was noted that this  
 36039 caused ambiguities because *tr* would not be able to tell whether adjacent octal sequences were  
 36040 intending to specify multi-byte characters or multiple single byte characters.  
 36041 IEEE Std 1003.1-2001 specifies that octal sequences always refer to single byte binary values.

#### 36042 **FUTURE DIRECTIONS**

36043 None.

#### 36044 **SEE ALSO**

36045 *sed*

#### 36046 **CHANGE HISTORY**

36047 First released in Issue 2.

#### 36048 **Issue 6**

36049 The `-C` operand is added, and the description of the `-c` operand is changed to align with the  
 36050 IEEE P1003.2b draft standard.

36051 The normative text is reworded to avoid use of the term “must” for application requirements.

36052 **NAME**

36053           true — return true value

36054 **SYNOPSIS**

36055           true

36056 **DESCRIPTION**36057           The *true* utility shall return with exit code zero.36058 **OPTIONS**

36059           None.

36060 **OPERANDS**

36061           None.

36062 **STDIN**

36063           Not used.

36064 **INPUT FILES**

36065           None.

36066 **ENVIRONMENT VARIABLES**

36067           None.

36068 **ASYNCHRONOUS EVENTS**

36069           Default.

36070 **STDOUT**

36071           Not used.

36072 **STDERR**

36073           Not used. |

36074 **OUTPUT FILES**

36075           None.

36076 **EXTENDED DESCRIPTION**

36077           None.

36078 **EXIT STATUS**

36079           Zero. |

36080 **CONSEQUENCES OF ERRORS**

36081           None.

36082 **APPLICATION USAGE**

36083           This utility is typically used in shell scripts, as shown in the EXAMPLES section. The special  
 36084           built-in utility `:` is sometimes more efficient than *true*.

36085 **EXAMPLES**

36086           This command is executed forever:

```
36087 while true
36088 do
36089 command
36090 done
```



36091 **RATIONALE**

36092           The *true* utility has been retained in this volume of IEEE Std 1003.1-2001, even though the shell  
36093           special built-in : provides similar functionality, because *true* is widely used in historical scripts  
36094           and is less cryptic to novice script readers.

36095 **FUTURE DIRECTIONS**

36096           None.

36097 **SEE ALSO**

36098           *false*, Section 2.9 (on page 47)

36099 **CHANGE HISTORY**

36100           First released in Issue 2.

36101 **Issue 6**

36102           IEEE Std 1003.1-2001 / Cor 1-2002, item XCU/TC1/D6/39 is applied, replacing the terms “None”  
36103           and “Default” from the STDERR and EXIT STATUS sections, respectively, with terms as defined  
36104           in Section 1.11 (on page 20).

36105 **NAME**

36106           tsort — topological sort

36107 **SYNOPSIS**36108 xSI       tsort [*file*]

36109

36110 **DESCRIPTION**36111           The *tsort* utility shall write to standard output a totally ordered list of items consistent with a  
36112           partial ordering of items contained in the input.36113           The application shall ensure that the input consists of pairs of items (non-empty strings)  
36114           separated by <blank>s. Pairs of different items indicate ordering. Pairs of identical items  
36115           indicate presence, but not ordering.36116 **OPTIONS**

36117           None.

36118 **OPERANDS**

36119           The following operand shall be supported:

36120           *file*           A pathname of a text file to order. If no *file* operand is given, the standard input  
36121           shall be used.36122 **STDIN**36123           The standard input shall be a text file that is used if no *file* operand is given.36124 **INPUT FILES**36125           The input file named by the *file* operand is a text file.36126 **ENVIRONMENT VARIABLES**36127           The following environment variables shall affect the execution of *tsort*:36128           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
36129                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
36130                           Internationalization Variables for the precedence of internationalization variables  
36131                           used to determine the values of locale categories.)36132           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
36133           internationalization variables.36134           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
36135           characters (for example, single-byte as opposed to multi-byte characters in  
36136           arguments and input files).36137           *LC\_MESSAGES*36138                           Determine the locale that should be used to affect the format and contents of  
36139           diagnostic messages written to standard error.36140           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.36141 **ASYNCHRONOUS EVENTS**

36142           Default.

36143 **STDOUT**36144           The standard output shall be a text file consisting of the order list produced from the partially  
36145           ordered input.

**36146 STDERR**

36147 The standard error shall be used only for diagnostic messages.

**36148 OUTPUT FILES**

36149 None.

**36150 EXTENDED DESCRIPTION**

36151 None.

**36152 EXIT STATUS**

36153 The following exit values shall be returned:

36154 0 Successful completion.

36155 >0 An error occurred.

**36156 CONSEQUENCES OF ERRORS**

36157 Default.

**36158 APPLICATION USAGE**

36159 The *LC\_COLLATE* variable need not affect the actions of *tsort*. The output ordering is not  
36160 lexicographic, but depends on the pairs of items given as input.

**36161 EXAMPLES**

36162 The command:

```
36163 tsort <<EOF
36164 a b c c d e
36165 g g
36166 f g e f
36167 h h
36168 EOF
```

36169 produces the output:

```
36170 a
36171 b
36172 c
36173 d
36174 e
36175 f
36176 g
36177 h
```

**36178 RATIONALE**

36179 None.

**36180 FUTURE DIRECTIONS**

36181 None.

**36182 SEE ALSO**

36183 None.

**36184 CHANGE HISTORY**

36185 First released in Issue 2.

**36186 Issue 6**

36187 The normative text is reworded to avoid use of the term “must” for application requirements.

36188 **NAME**

36189           tty — return user's terminal name

36190 **SYNOPSIS**

36191           tty

36192 **DESCRIPTION**

36193           The *tty* utility shall write to the standard output the name of the terminal that is open as  
 36194           standard input. The name that is used shall be equivalent to the string that would be returned by  
 36195           the *ttyname()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001.

36196 **OPTIONS**

36197           The *tty* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
 36198           Utility Syntax Guidelines.

36199 **OPERANDS**

36200           None.

36201 **STDIN**

36202           While no input is read from standard input, standard input shall be examined to determine  
 36203           whether or not it is a terminal, and, if so, to determine the name of the terminal.

36204 **INPUT FILES**

36205           None.

36206 **ENVIRONMENT VARIABLES**36207           The following environment variables shall affect the execution of *tty*:

36208           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 36209                        (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 36210                        Internationalization Variables for the precedence of internationalization variables  
 36211                        used to determine the values of locale categories.)

36212           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
 36213                        internationalization variables.

36214           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 36215                        characters (for example, single-byte as opposed to multi-byte characters in  
 36216                        arguments).

36217           *LC\_MESSAGES*

36218                        Determine the locale that should be used to affect the format and contents of  
 36219                        diagnostic messages written to standard error and informative messages written to  
 36220                        standard output.

36221 xSI       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36222 **ASYNCHRONOUS EVENTS**

36223           Default.

36224 **STDOUT**

36225           If standard input is a terminal device, a pathname of the terminal as specified by the *ttyname()*  
 36226           function defined in the System Interfaces volume of IEEE Std 1003.1-2001 shall be written in the  
 36227           following format:

36228           "%s\n", &lt;terminal name&gt;

36229           Otherwise, a message shall be written indicating that standard input is not connected to a  
 36230           terminal. In the POSIX locale, the *tty* utility shall use the format:

36231 "not a tty\n"

36232 **STDERR**

36233 The standard error shall be used only for diagnostic messages.

36234 **OUTPUT FILES**

36235 None.

36236 **EXTENDED DESCRIPTION**

36237 None.

36238 **EXIT STATUS**

36239 The following exit values shall be returned:

36240 0 Standard input is a terminal.

36241 1 Standard input is not a terminal.

36242 >1 An error occurred.

36243 **CONSEQUENCES OF ERRORS**

36244 Default.

36245 **APPLICATION USAGE**

36246 This utility checks the status of the file open as standard input against that of an  
36247 implementation-defined set of files. It is possible that no match can be found, or that the match  
36248 found need not be the same file as that which was opened for standard input (although they are  
36249 the same device).

36250 **EXAMPLES**

36251 None.

36252 **RATIONALE**

36253 None.

36254 **FUTURE DIRECTIONS**

36255 None.

36256 **SEE ALSO**

36257 The System Interfaces volume of IEEE Std 1003.1-2001, *isatty()*, *ttyname()*

36258 **CHANGE HISTORY**

36259 First released in Issue 2.

36260 **Issue 5**

36261 The SYNOPSIS is changed to indicate two forms of the command, with the second form marked  
36262 as obsolete. This is a clarification and does not change the functionality published in previous  
36263 issues.

36264 **Issue 6**

36265 The obsolescent **-s** option is removed.

36266 **NAME**

36267           type — write a description of command type

36268 **SYNOPSIS**

36269 xSI       type name...

36270

36271 **DESCRIPTION**36272       The *type* utility shall indicate how each argument would be interpreted if used as a command  
36273       name.36274 **OPTIONS**

36275       None.

36276 **OPERANDS**

36277       The following operand shall be supported:

36278       *name*           A name to be interpreted.36279 **STDIN**

36280       Not used.

36281 **INPUT FILES**

36282       None.

36283 **ENVIRONMENT VARIABLES**36284       The following environment variables shall affect the execution of *type*:36285       *LANG*           Provide a default value for the internationalization variables that are unset or null.  
36286                       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
36287                       Internationalization Variables for the precedence of internationalization variables  
36288                       used to determine the values of locale categories.)36289       *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
36290       internationalization variables.36291       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
36292       characters (for example, single-byte as opposed to multi-byte characters in  
36293       arguments).36294       *LC\_MESSAGES*   Determine the locale that should be used to affect the format and contents of  
36295       diagnostic messages written to standard error.  
3629636297       *NLSPATH*       Determine the location of message catalogs for the processing of *LC\_MESSAGES*.36298       *PATH*           Determine the location of *name*, as described in the Base Definitions volume of  
36299       IEEE Std 1003.1-2001, Chapter 8, Environment Variables.36300 **ASYNCHRONOUS EVENTS**

36301       Default.

36302 **STDOUT**36303       The standard output of *type* contains information about each operand in an unspecified format.  
36304       The information provided typically identifies the operand as a shell built-in, function, alias, or  
36305       keyword, and where applicable, may display the operand's pathname.

36306 **STDERR**

36307           The standard error shall be used only for diagnostic messages.

36308 **OUTPUT FILES**

36309           None.

36310 **EXTENDED DESCRIPTION**

36311           None.

36312 **EXIT STATUS**

36313           The following exit values shall be returned:

36314           0   Successful completion.

36315           >0  An error occurred.

36316 **CONSEQUENCES OF ERRORS**

36317           Default.

36318 **APPLICATION USAGE**

36319           Since *type* must be aware of the contents of the current shell execution environment (such as the  
36320           lists of commands, functions, and built-ins processed by *hash*), it is always provided as a shell  
36321           regular built-in. If it is called in a separate utility execution environment, such as one of the  
36322           following:

36323           nohup *type* writer

36324           find . -type f | xargs *type*

36325           it might not produce accurate results.

36326 **EXAMPLES**

36327           None.

36328 **RATIONALE**

36329           None.

36330 **FUTURE DIRECTIONS**

36331           None.

36332 **SEE ALSO**

36333           *command*, *hash*

36334 **CHANGE HISTORY**

36335           First released in Issue 2.

36336 **NAME**

36337           ulimit — set or report file size limit

36338 **SYNOPSIS**36339 xSI       ulimit [-f] [*blocks*]

36340

36341 **DESCRIPTION**

36342       The *ulimit* utility shall set or report the file-size writing limit imposed on files written by the  
 36343       shell and its child processes (files of any size may be read). Only a process with appropriate  
 36344       privileges can increase the limit.

36345 **OPTIONS**

36346       The *ulimit* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 36347       12.2, Utility Syntax Guidelines.

36348       The following option shall be supported:

36349       **-f**           Set (or report, if no *blocks* operand is present), the file size limit in blocks. The **-f**  
 36350       option shall also be the default case.

36351 **OPERANDS**

36352       The following operand shall be supported:

36353       *blocks*       The number of 512-byte blocks to use as the new file size limit.

36354 **STDIN**

36355       Not used.

36356 **INPUT FILES**

36357       None.

36358 **ENVIRONMENT VARIABLES**

36359       The following environment variables shall affect the execution of *ulimit*:

36360       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 36361       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 36362       Internationalization Variables for the precedence of internationalization variables  
 36363       used to determine the values of locale categories.)

36364       *LC\_ALL*      If set to a non-empty string value, override the values of all the other  
 36365       internationalization variables.

36366       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 36367       characters (for example, single-byte as opposed to multi-byte characters in  
 36368       arguments).

36369       *LC\_MESSAGES*   Determine the locale that should be used to affect the format and contents of  
 36370       diagnostic messages written to standard error.  
 36371

36372       *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36373 **ASYNCHRONOUS EVENTS**

36374       Default.

36375 **STDOUT**

36376       The standard output shall be used when no *blocks* operand is present. If the current number of  
 36377       blocks is limited, the number of blocks in the current limit shall be written in the following  
 36378       format:



36379 "%d\n", <number of 512-byte blocks>

36380 If there is no current limit on the number of blocks, in the POSIX locale the following format  
36381 shall be used:

36382 "unlimited\n"

#### 36383 **STDERR**

36384 The standard error shall be used only for diagnostic messages.

#### 36385 **OUTPUT FILES**

36386 None.

#### 36387 **EXTENDED DESCRIPTION**

36388 None.

#### 36389 **EXIT STATUS**

36390 The following exit values shall be returned:

36391 0 Successful completion.

36392 >0 A request for a higher limit was rejected or an error occurred.

#### 36393 **CONSEQUENCES OF ERRORS**

36394 Default.

#### 36395 **APPLICATION USAGE**

36396 Since *ulimit* affects the current shell execution environment, it is always provided as a shell  
36397 regular built-in. If it is called in a separate utility execution environment, such as one of the  
36398 following:

36399 nohup ulimit -f 10000

36400 env ulimit 10000

36401 it does not affect the file size limit of the caller's environment.

36402 Once a limit has been decreased by a process, it cannot be increased (unless appropriate  
36403 privileges are involved), even back to the original system limit.

#### 36404 **EXAMPLES**

36405 Set the file size limit to 51 200 bytes:

36406 ulimit -f 100

#### 36407 **RATIONALE**

36408 None.

#### 36409 **FUTURE DIRECTIONS**

36410 None.

#### 36411 **SEE ALSO**

36412 The System Interfaces volume of IEEE Std 1003.1-2001, *ulimit()*

#### 36413 **CHANGE HISTORY**

36414 First released in Issue 2.

36415 **NAME**

36416           umask — get or set the file mode creation mask

36417 **SYNOPSIS**36418           umask [-S] [*mask*]36419 **DESCRIPTION**

36420           The *umask* utility shall set the file mode creation mask of the current shell execution  
 36421           environment (see Section 2.12 (on page 61)) to the value specified by the *mask* operand. This  
 36422           mask shall affect the initial value of the file permission bits of subsequently created files. If *umask*  
 36423           is called in a subshell or separate utility execution environment, such as one of the following:

36424           (*umask* 002)  
 36425           nohup *umask* ...  
 36426           find . -exec *umask* ... \;

36427           it shall not affect the file mode creation mask of the caller's environment.

36428           If the *mask* operand is not specified, the *umask* utility shall write to standard output the value of  
 36429           the invoking process' file mode creation mask.

36430 **OPTIONS**

36431           The *umask* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 36432           12.2, Utility Syntax Guidelines.

36433           The following option shall be supported:

36434           -S           Produce symbolic output.

36435           The default output style is unspecified, but shall be recognized on a subsequent invocation of  
 36436           *umask* on the same system as a *mask* operand to restore the previous file mode creation mask.

36437 **OPERANDS**

36438           The following operand shall be supported:

36439           *mask*           A string specifying the new file mode creation mask. The string is treated in the  
 36440           same way as the *mode* operand described in the EXTENDED DESCRIPTION  
 36441           section for *chmod*.

36442           For a *symbolic\_mode* value, the new value of the file mode creation mask shall be  
 36443           the logical complement of the file permission bits portion of the file mode specified  
 36444           by the *symbolic\_mode* string.

36445           In a *symbolic\_mode* value, the permissions *op* characters '+' and '-' shall be  
 36446           interpreted relative to the current file mode creation mask; '+' shall cause the bits  
 36447           for the indicated permissions to be cleared in the mask; '-' shall cause the bits for  
 36448           the indicated permissions to be set in the mask.

36449           The interpretation of *mode* values that specify file mode bits other than the file  
 36450           permission bits is unspecified.

36451           In the octal integer form of *mode*, the specified bits are set in the file mode creation  
 36452           mask.

36453           The file mode creation mask shall be set to the resulting numeric value.

36454           The default output of a prior invocation of *umask* on the same system with no  
 36455           operand also shall be recognized as a *mask* operand.

36456 **STDIN**

36457 Not used.

36458 **INPUT FILES**

36459 None.

36460 **ENVIRONMENT VARIABLES**36461 The following environment variables shall affect the execution of *umask*:

36462 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36463 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 36464 Internationalization Variables for the precedence of internationalization variables  
 36465 used to determine the values of locale categories.)

36466 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36467 internationalization variables.

36468 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36469 characters (for example, single-byte as opposed to multi-byte characters in  
 36470 arguments).

36471 *LC\_MESSAGES*

36472 Determine the locale that should be used to affect the format and contents of  
 36473 diagnostic messages written to standard error.

36474 *XS* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36475 **ASYNCHRONOUS EVENTS**

36476 Default.

36477 **STDOUT**

36478 When the *mask* operand is not specified, the *umask* utility shall write a message to standard  
 36479 output that can later be used as a *umask mask* operand.

36480 If *-S* is specified, the message shall be in the following format:

36481 "u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,  
 36482 <other permissions>

36483 where the three values shall be combinations of letters from the set {r, w, x}; the presence of a  
 36484 letter shall indicate that the corresponding bit is clear in the file mode creation mask.

36485 If a *mask* operand is specified, there shall be no output written to standard output.36486 **STDERR**

36487 The standard error shall be used only for diagnostic messages.

36488 **OUTPUT FILES**

36489 None.

36490 **EXTENDED DESCRIPTION**

36491 None.

36492 **EXIT STATUS**

36493 The following exit values shall be returned:

36494 0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

36495 &gt;0 An error occurred.

36496 **CONSEQUENCES OF ERRORS**

36497 Default.

36498 **APPLICATION USAGE**

36499 Since *umask* affects the current shell execution environment, it is generally provided as a shell  
 36500 regular built-in.

36501 In contrast to the negative permission logic provided by the file mode creation mask and the  
 36502 octal number form of the *mask* argument, the symbolic form of the *mask* argument specifies those  
 36503 permissions that are left alone.

36504 **EXAMPLES**

36505 Either of the commands:

36506 `umask a=rx,ug+w`36507 `umask 002`

36508 sets the mode mask so that subsequently created files have their S\_IWOTH bit cleared.

36509 After setting the mode mask with either of the above commands, the *umask* command can be  
 36510 used to write out the current value of the mode mask:

36511 `$ umask`36512 `0002`

36513 (The output format is unspecified, but historical implementations use the octal integer mode  
 36514 format.)

36515 `$ umask -S`36516 `u=rwx,g=rwx,o=rx`

36517 Either of these outputs can be used as the mask operand to a subsequent invocation of the *umask*  
 36518 utility.

36519 Assuming the mode mask is set as above, the command:

36520 `umask g-w`

36521 sets the mode mask so that subsequently created files have their S\_IWGRP and S\_IWOTH bits  
 36522 cleared.

36523 The command:

36524 `umask -- -w`

36525 sets the mode mask so that subsequently created files have all their write bits cleared. Note that  
 36526 *mask* operands `-r`, `-w`, `-x` or anything beginning with a hyphen, must be preceded by `--` to  
 36527 keep it from being interpreted as an option.

36528 **RATIONALE**

36529 Since *umask* affects the current shell execution environment, it is generally provided as a shell  
 36530 regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
 36531 of the following:

36532 `(umask 002)`36533 `nohup umask ...`36534 `find . -exec umask ... \;`

36535 it does not affect the file mode creation mask of the environment of the caller.

36536 The description of the historical utility was modified to allow it to use the symbolic modes of  
 36537 *chmod*. The `-s` option used in early proposals was changed to `-S` because `-s` could be confused

36538 with a *symbolic\_mode* form of mask referring to the S\_ISUID and S\_ISGID bits.

36539 The default output style is implementation-defined to permit implementors to provide  
36540 migration to the new symbolic style at the time most appropriate to their users. A **-o** flag to  
36541 force octal mode output was omitted because the octal mode may not be sufficient to specify all  
36542 of the information that may be present in the file mode creation mask when more secure file  
36543 access permission checks are implemented.

36544 It has been suggested that trusted systems developers might appreciate ameliorating the  
36545 requirement that the mode mask “affects” the file access permissions, since it seems access  
36546 control lists might replace the mode mask to some degree. The wording has been changed to say  
36547 that it affects the file permission bits, and it leaves the details of the behavior of how they affect  
36548 the file access permissions to the description in the System Interfaces volume of  
36549 IEEE Std 1003.1-2001.

36550 **FUTURE DIRECTIONS**

36551 None.

36552 **SEE ALSO**

36553 Chapter 2 (on page 29), *chmod*, the System Interfaces volume of IEEE Std 1003.1-2001, *umask()*

36554 **CHANGE HISTORY**

36555 First released in Issue 2.

36556 **Issue 6**

36557 The following new requirements on POSIX implementations derive from alignment with the  
36558 Single UNIX Specification:

- 36559
- The octal mode is supported.

36560 **NAME**

36561 unalias — remove alias definitions

36562 **SYNOPSIS**36563 UP unalias *alias-name*...

36564 unalias -a

36565

36566 **DESCRIPTION**

36567 The *unalias* utility shall remove the definition for each alias name specified. See Section 2.3.1 (on  
 36568 page 32). The aliases shall be removed from the current shell execution environment; see Section  
 36569 2.12 (on page 61).

36570 **OPTIONS**

36571 The *unalias* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 36572 12.2, Utility Syntax Guidelines.

36573 The following option shall be supported:

36574 **-a** Remove all alias definitions from the current shell execution environment.

36575 **OPERANDS**

36576 The following operand shall be supported:

36577 *alias-name* The name of an alias to be removed.

36578 **STDIN**

36579 Not used.

36580 **INPUT FILES**

36581 None.

36582 **ENVIRONMENT VARIABLES**

36583 The following environment variables shall affect the execution of *unalias*:

36584 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 36585 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 36586 Internationalization Variables for the precedence of internationalization variables  
 36587 used to determine the values of locale categories.)

36588 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 36589 internationalization variables.

36590 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 36591 characters (for example, single-byte as opposed to multi-byte characters in  
 36592 arguments).

36593 **LC\_MESSAGES**

36594 Determine the locale that should be used to affect the format and contents of  
 36595 diagnostic messages written to standard error.

36596 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36597 **ASYNCHRONOUS EVENTS**

36598 Default.

36599 **STDOUT**

36600 Not used.

36601 **STDERR**

36602           The standard error shall be used only for diagnostic messages.

36603 **OUTPUT FILES**

36604           None.

36605 **EXTENDED DESCRIPTION**

36606           None.

36607 **EXIT STATUS**

36608           The following exit values shall be returned:

36609           0   Successful completion.

36610           >0   One of the *alias-name* operands specified did not represent a valid alias definition, or an  
36611           error occurred.

36612 **CONSEQUENCES OF ERRORS**

36613           Default.

36614 **APPLICATION USAGE**

36615           Since *unalias* affects the current shell execution environment, it is generally provided as a shell  
36616           regular built-in.

36617 **EXAMPLES**

36618           None.

36619 **RATIONALE**

36620           The *unalias* description is based on that from historical KornShell implementations. Known  
36621           differences exist between that and the C shell. The KornShell version was adopted to be  
36622           consistent with all the other KornShell features in this volume of IEEE Std 1003.1-2001, such as  
36623           command line editing.

36624           The *-a* option is the equivalent of the *unalias \** form of the C shell and is provided to address  
36625           security concerns about unknown aliases entering the environment of a user (or application)  
36626           through the allowable implementation-defined predefined alias route or as a result of an *ENV*  
36627           file. (Although *unalias* could be used to simplify the “secure” shell script shown in the *command*  
36628           rationale, it does not obviate the need to quote all command names. An initial call to *unalias -a*  
36629           would have to be quoted in case there was an alias for *unalias*.)

36630 **FUTURE DIRECTIONS**

36631           None.

36632 **SEE ALSO**

36633           Chapter 2 (on page 29), *alias*

36634 **CHANGE HISTORY**

36635           First released in Issue 4.

36636 **Issue 6**

36637           This utility is marked as part of the User Portability Utilities option.

36638 **NAME**

36639            uname — return system name

36640 **SYNOPSIS**

36641            uname [-snrvma]

36642 **DESCRIPTION**

36643            By default, the *uname* utility shall write the operating system name to standard output. When  
 36644            options are specified, symbols representing one or more system characteristics shall be written  
 36645            to the standard output. The format and contents of the symbols are implementation-defined. On  
 36646            systems conforming to the System Interfaces volume of IEEE Std 1003.1-2001, the symbols  
 36647            written shall be those supported by the *uname()* function as defined in the System Interfaces  
 36648            volume of IEEE Std 1003.1-2001.

36649 **OPTIONS**

36650            The *uname* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 36651            12.2, Utility Syntax Guidelines.

36652            The following options shall be supported:

- 36653            **-a**            Behave as though all of the options **-mnrsv** were specified.
- 36654            **-m**            Write the name of the hardware type on which the system is running to standard  
 36655            output.
- 36656            **-n**            Write the name of this node within an implementation-defined communications  
 36657            network.
- 36658            **-r**            Write the current release level of the operating system implementation.
- 36659            **-s**            Write the name of the implementation of the operating system.
- 36660            **-v**            Write the current version level of this release of the operating system  
 36661            implementation.
- 36662            If no options are specified, the *uname* utility shall write the operating system name, as if the **-s**  
 36663            option had been specified.

36664 **OPERANDS**

36665            None.

36666 **STDIN**

36667            Not used.

36668 **INPUT FILES**

36669            None.

36670 **ENVIRONMENT VARIABLES**36671            The following environment variables shall affect the execution of *uname*:

- 36672            **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 36673            (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 36674            Internationalization Variables for the precedence of internationalization variables  
 36675            used to determine the values of locale categories.)
- 36676            **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 36677            internationalization variables.
- 36678            **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
 36679            characters (for example, single-byte as opposed to multi-byte characters in  
 36680            arguments).



36681 **LC\_MESSAGES**  
 36682 Determine the locale that should be used to affect the format and contents of  
 36683 diagnostic messages written to standard error.

36684 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

36685 **ASYNCHRONOUS EVENTS**  
 36686 Default.

36687 **STDOUT**  
 36688 By default, the output shall be a single line of the following form:  
 36689 "%s\n", <sysname>  
 36690 If the **-a** option is specified, the output shall be a single line of the following form:  
 36691 "%s %s %s %s %s\n", <sysname>, <nodename>, <release>,  
 36692 <version>, <machine>  
 36693 Additional implementation-defined symbols may be written; all such symbols shall be written at  
 36694 the end of the line of output before the <newline>.  
 36695 If options are specified to select different combinations of the symbols, only those symbols shall  
 36696 be written, in the order shown above for the **-a** option. If a symbol is not selected for writing, its  
 36697 corresponding trailing <blank>s also shall not be written.

36698 **STDERR**  
 36699 The standard error shall be used only for diagnostic messages.

36700 **OUTPUT FILES**  
 36701 None.

36702 **EXTENDED DESCRIPTION**  
 36703 None.

36704 **EXIT STATUS**  
 36705 The following exit values shall be returned:  
 36706 0 The requested information was successfully written.  
 36707 >0 An error occurred.

36708 **CONSEQUENCES OF ERRORS**  
 36709 Default.

36710 **APPLICATION USAGE**  
 36711 Note that any of the symbols could include embedded <space>s, which may affect parsing  
 36712 algorithms if multiple options are selected for output.  
 36713 The node name is typically a name that the system uses to identify itself for inter-system  
 36714 communication addressing.

36715 **EXAMPLES**  
 36716 The following command:  
 36717 `uname -sr`  
 36718 writes the operating system name and release level, separated by one or more <blank>s.

36719 **RATIONALE**

36720 It was suggested that this utility cannot be used portably since the format of the symbols is  
36721 implementation-defined. The POSIX.1 working group could not achieve consensus on defining  
36722 these formats in the underlying *uname()* function, and there was no expectation that this volume  
36723 of IEEE Std 1003.1-2001 would be any more successful. Some applications may still find this  
36724 historical utility of value. For example, the symbols could be used for system log entries or for  
36725 comparison with operator or user input.

36726 **FUTURE DIRECTIONS**

36727 None.

36728 **SEE ALSO**

36729 The System Interfaces volume of IEEE Std 1003.1-2001, *uname()*

36730 **CHANGE HISTORY**

36731 First released in Issue 2.

36732 **NAME**

36733 uncompress — expand compressed data

36734 **SYNOPSIS**36735 xSI uncompress [-cfv] [*file...*]

36736

36737 **DESCRIPTION**

36738 The *uncompress* utility shall restore files to their original state after they have been compressed  
 36739 using the *compress* utility. If no files are specified, the standard input shall be uncompressed to  
 36740 the standard output. If the invoking process has appropriate privileges, the ownership, modes,  
 36741 access time, and modification time of the original file shall be preserved.

36742 This utility shall support the uncompressing of any files produced by the *compress* utility on the  
 36743 same implementation. For files produced by *compress* on other systems, *uncompress* supports 9 to  
 36744 14-bit compression (see *compress*, **-b**); it is implementation-defined whether values of **-b** greater  
 36745 than 14 are supported.

36746 **OPTIONS**

36747 The *uncompress* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001,  
 36748 Section 12.2, Utility Syntax Guidelines.

36749 The following options shall be supported:

- 36750 **-c** Write to standard output; no files are changed.
- 36751 **-f** Do not prompt for overwriting files. Except when run in the background, if **-f** is  
 36752 not given the user shall be prompted as to whether an existing file should be  
 36753 overwritten. If the standard input is not a terminal and **-f** is not given, *uncompress*  
 36754 shall write a diagnostic message to standard error and exit with a status greater  
 36755 than zero.
- 36756 **-v** Write messages to standard error concerning the expansion of each file.

36757 **OPERANDS**

36758 The following operand shall be supported:

- 36759 *file* A pathname of a file. If *file* already has the **.Z** suffix specified, it shall be used as the  
 36760 input file and the output file shall be named **file** with the **.Z** suffix removed.  
 36761 Otherwise, *file* shall be used as the name of the output file and **file** with the **.Z**  
 36762 suffix appended shall be used as the input file.

36763 **STDIN**

36764 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is ' - '.

36765 **INPUT FILES**

36766 Input files shall be in the format produced by the *compress* utility.

36767 **ENVIRONMENT VARIABLES**

36768 The following environment variables shall affect the execution of *uncompress*:

- 36769 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 36770 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 36771 Internationalization Variables for the precedence of internationalization variables  
 36772 used to determine the values of locale categories.)
- 36773 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 36774 internationalization variables.

36775        *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
36776                   characters (for example, single-byte as opposed to multi-byte characters in  
36777                   arguments).

36778        *LC\_MESSAGES*  
36779                   Determine the locale that should be used to affect the format and contents of  
36780                   diagnostic messages written to standard error.

36781        *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36782 **ASYNCHRONOUS EVENTS**

36783        Default.

36784 **STDOUT**

36785        When there are no *file* operands or the **-c** option is specified, the uncompressed output is written  
36786        to standard output.

36787 **STDERR**

36788        Prompts shall be written to the standard error output under the conditions specified in the  
36789        DESCRIPTION and OPTIONS sections. The prompts shall contain the *file* pathname, but their  
36790        format is otherwise unspecified. Otherwise, the standard error output shall be used only for  
36791        diagnostic messages.

36792 **OUTPUT FILES**

36793        Output files are the same as the respective input files to *compress*.

36794 **EXTENDED DESCRIPTION**

36795        None.

36796 **EXIT STATUS**

36797        The following exit values shall be returned:

36798        0   Successful completion.

36799        >0   An error occurred.

36800 **CONSEQUENCES OF ERRORS**

36801        The input file remains unmodified.

36802 **APPLICATION USAGE**

36803        The limit of 14 on the *compress -b bits* argument is to achieve portability to all systems (within  
36804        the restrictions imposed by the lack of an explicit published file format). Some implementations  
36805        based on 16-bit architectures cannot support 15 or 16-bit uncompression.

36806 **EXAMPLES**

36807        None.

36808 **RATIONALE**

36809        None.

36810 **FUTURE DIRECTIONS**

36811        None.

36812 **SEE ALSO**

36813        *compress*, *zcat*

36814 **CHANGE HISTORY**

36815        First released in Issue 4.

36816 **Issue 6**

36817 The normative text is reworded to avoid use of the term “must” for application requirements.

36818 **NAME**

36819 unexpand — convert spaces to tabs

36820 **SYNOPSIS**36821 UP unexpand [ -a | -t *tablist*] [*file...*]

36822

36823 **DESCRIPTION**

36824 The *unexpand* utility shall copy files or standard input to standard output, converting <blank>s  
 36825 at the beginning of each line into the maximum number of <tab>s followed by the minimum  
 36826 number of <space>s needed to fill the same column positions originally filled by the translated  
 36827 <blank>s. By default, tabstops shall be set at every eighth column position. Each <backspace>  
 36828 shall be copied to the output, and shall cause the column position count for tab calculations to be  
 36829 decremented; the count shall never be decremented to a value less than one.

36830 **OPTIONS**

36831 The *unexpand* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001,  
 36832 Section 12.2, Utility Syntax Guidelines.

36833 The following options shall be supported:

36834 **-a** In addition to translating <blank>s at the beginning of each line, translate all  
 36835 sequences of two or more <blank>s immediately preceding a tab stop to the  
 36836 maximum number of <tab>s followed by the minimum number of <space>s  
 36837 needed to fill the same column positions originally filled by the translated  
 36838 <blank>s.

36839 **-t *tablist*** Specify the tab stops. The application shall ensure that the *tablist* option-argument  
 36840 is a single argument consisting of a single positive decimal integer or multiple  
 36841 positive decimal integers, separated by <blank>s or commas, in ascending order. If  
 36842 a single number is given, tabs shall be set *tablist* column positions apart instead of  
 36843 the default 8. If multiple numbers are given, the tabs shall be set at those specific  
 36844 column positions.

36845 The application shall ensure that each tab-stop position *N* is an integer value  
 36846 greater than zero, and the list shall be in strictly ascending order. This is taken to  
 36847 mean that, from the start of a line of output, tabbing to position *N* shall cause the  
 36848 next character output to be in the (*N*+1)th column position on that line. When the  
 36849 **-t** option is not specified, the default shall be the equivalent of specifying **-t 8**  
 36850 (except for the interaction with **-a**, described below).

36851 No <space>-to-<tab> conversions shall occur for characters at positions beyond  
 36852 the last of those specified in a multiple tab-stop list.

36853 When **-t** is specified, the presence or absence of the **-a** option shall be ignored;  
 36854 conversion shall not be limited to the processing of leading <blank>s.

36855 **OPERANDS**

36856 The following operand shall be supported:

36857 *file* A pathname of a text file to be used as input.

36858 **STDIN**

36859 See the INPUT FILES section.

36860 **INPUT FILES**

36861 The input files shall be text files.

36862 **ENVIRONMENT VARIABLES**36863 The following environment variables shall affect the execution of *unexpand*:

36864 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36865 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 36866 Internationalization Variables for the precedence of internationalization variables  
 36867 used to determine the values of locale categories.)

36868 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36869 internationalization variables.

36870 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36871 characters (for example, single-byte as opposed to multi-byte characters in  
 36872 arguments and input files), the processing of <tab>s and <space>s, and for the  
 36873 determination of the width in column positions each character would occupy on  
 36874 an output device.

36875 *LC\_MESSAGES*  
 36876 Determine the locale that should be used to affect the format and contents of  
 36877 diagnostic messages written to standard error.

36878 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36879 **ASYNCHRONOUS EVENTS**

36880 Default.

36881 **STDOUT**

36882 The standard output shall be equivalent to the input files with the specified <space>-to-<tab>  
 36883 conversions.

36884 **STDERR**

36885 The standard error shall be used only for diagnostic messages.

36886 **OUTPUT FILES**

36887 None.

36888 **EXTENDED DESCRIPTION**

36889 None.

36890 **EXIT STATUS**

36891 The following exit values shall be returned:

36892 0 Successful completion.

36893 &gt;0 An error occurred.

36894 **CONSEQUENCES OF ERRORS**

36895 Default.

**36896 APPLICATION USAGE**

36897 One non-intuitive aspect of *unexpand* is its restriction to leading spaces when neither **-a** nor **-t** is  
36898 specified. Users who always want to convert all spaces in a file can easily alias *unexpand* to use  
36899 the **-a** or **-t 8** option.

**36900 EXAMPLES**

36901 None.

**36902 RATIONALE**

36903 On several occasions, consideration was given to adding a **-t** option to the *unexpand* utility to  
36904 complement the **-t** in *expand* (see *expand*). The historical intent of *unexpand* was to translate  
36905 multiple <blank>s into tab stops, where tab stops were a multiple of eight column positions on  
36906 most UNIX systems. An early proposal omitted **-t** because it seemed outside the scope of the  
36907 User Portability Utilities option; it was not described in any of the base documents. However,  
36908 hard-coding tab stops every eight columns was not suitable for the international community and  
36909 broke historical precedents for some vendors in the FORTRAN community, so **-t** was restored  
36910 in conjunction with the list of valid extension categories considered by the standard developers.  
36911 Thus, *unexpand* is now the logical converse of *expand*.

**36912 FUTURE DIRECTIONS**

36913 None.

**36914 SEE ALSO**

36915 *expand*, *tabs*

**36916 CHANGE HISTORY**

36917 First released in Issue 4.

**36918 Issue 6**

36919 This utility is marked as part of the User Portability Utilities option.

36920 The definition of the *LC\_CTYPE* environment variable is changed to align with the  
36921 IEEE P1003.2b draft standard.

36922 The normative text is reworded to avoid use of the term “must” for application requirements.



36923 **NAME**36924 unget — undo a previous get of an SCCS file (**DEVELOPMENT**)36925 **SYNOPSIS**36926 xSI unget [-ns] [-r *SID*] *file...*

36927

36928 **DESCRIPTION**36929 The *unget* utility shall reverse the effect of a *get -e* done prior to creating the intended new delta.36930 **OPTIONS**36931 The *unget* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
36932 12.2, Utility Syntax Guidelines.

36933 The following options shall be supported:

36934 **-r *SID*** Uniquely identify which delta is no longer intended. (This would have been  
36935 specified by *get* as the new delta.) The use of this option is necessary only if two or  
36936 more outstanding *get* commands for editing on the same SCCS file were done by  
36937 the same person (login name).36938 **-s** Suppress the writing to standard output of the intended delta's *SID*.36939 **-n** Retain the file that was obtained by *get*, which would normally be removed from  
36940 the current directory.36941 **OPERANDS**

36942 The following operands shall be supported:

36943 ***file*** A pathname of an existing SCCS file or a directory. If *file* is a directory, the *unget*  
36944 utility shall behave as though each file in the directory were specified as a named  
36945 file, except that non-SCCS files (last component of the pathname does not begin  
36946 with **s**.) and unreadable files shall be silently ignored.36947 If exactly one *file* operand appears, and it is **'-'**, the standard input shall be read;  
36948 each line of the standard input shall be taken to be the name of an SCCS file to be  
36949 processed. Non-SCCS files and unreadable files shall be silently ignored.36950 **STDIN**36951 The standard input shall be a text file used only when the *file* operand is specified as **'-'**. Each  
36952 line of the text file shall be interpreted as an SCCS pathname.36953 **INPUT FILES**

36954 Any SCCS files processed shall be files of an unspecified format.

36955 **ENVIRONMENT VARIABLES**36956 The following environment variables shall affect the execution of *unget*:36957 ***LANG*** Provide a default value for the internationalization variables that are unset or null.  
36958 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
36959 Internationalization Variables for the precedence of internationalization variables  
36960 used to determine the values of locale categories.)36961 ***LC\_ALL*** If set to a non-empty string value, override the values of all the other  
36962 internationalization variables.36963 ***LC\_CTYPE*** Determine the locale for the interpretation of sequences of bytes of text data as  
36964 characters (for example, single-byte as opposed to multi-byte characters in  
36965 arguments and input files).

36966 **LC\_MESSAGES**

36967 Determine the locale that should be used to affect the format and contents of  
36968 diagnostic messages written to standard error.

36969 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36970 **ASYNCHRONOUS EVENTS**

36971 Default.

36972 **STDOUT**

36973 The standard output shall consist of a line for each file, in the following format:

36974 "%s\n", <*SID removed from file*>

36975 If there is more than one named file or if a directory or standard input is named, each pathname  
36976 shall be written before each of the preceding lines:

36977 "\n%s:\n", <*pathname*>

36978 **STDERR**

36979 The standard error shall be used only for diagnostic messages.

36980 **OUTPUT FILES**

36981 Any SCCS files updated shall be files of an unspecified format. During processing of a *file*, a  
36982 locking *z-file*, as described in *get*, and a *q-file* (a working copy of the *p-file*), may be created and  
36983 deleted. The *p-file* and *g-file*, as described in *get*, shall be deleted.

36984 **EXTENDED DESCRIPTION**

36985 None.

36986 **EXIT STATUS**

36987 The following exit values shall be returned:

36988 0 Successful completion.

36989 >0 An error occurred.

36990 **CONSEQUENCES OF ERRORS**

36991 Default.

36992 **APPLICATION USAGE**

36993 None.

36994 **EXAMPLES**

36995 None.

36996 **RATIONALE**

36997 None.

36998 **FUTURE DIRECTIONS**

36999 None.

37000 **SEE ALSO**

37001 *delta*, *get*, *sact*

37002 **CHANGE HISTORY**

37003 First released in Issue 2.

37004 **Issue 6**

37005 The normative text is reworded to avoid use of the term “must” for application requirements.

37006 **NAME**37007            `uniq` — report or filter out repeated lines in a file37008 **SYNOPSIS**37009            `uniq [-c|-d|-u] [-f fields] [-s char] [input_file [output_file]]`37010 **DESCRIPTION**

37011        The *uniq* utility shall read an input file comparing adjacent lines, and write one copy of each  
 37012        input line on the output. The second and succeeding copies of repeated adjacent input lines shall  
 37013        not be written.

37014        Repeated lines in the input shall not be detected if they are not adjacent.

37015 **OPTIONS**

37016        The *uniq* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 37017        12.2, Utility Syntax Guidelines.

37018        The following options shall be supported:

37019        **-c**            Precede each output line with a count of the number of times the line occurred in  
 37020                        the input.

37021        **-d**            Suppress the writing of lines that are not repeated in the input.

37022        **-f *fields***    Ignore the first *fields* fields on each input line when doing comparisons, where  
 37023                        *fields* is a positive decimal integer. A field is the maximal string matched by the  
 37024                        basic regular expression:

37025                        `[[:blank:]]* [^[[:blank:]]]*`

37026                        If the *fields* option-argument specifies more fields than appear on an input line, a  
 37027                        null string shall be used for comparison.

37028        **-s *chars***    Ignore the first *chars* characters when doing comparisons, where *chars* shall be a  
 37029                        positive decimal integer. If specified in conjunction with the **-f** option, the first  
 37030                        *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-  
 37031                        argument specifies more characters than remain on an input line, a null string shall  
 37032                        be used for comparison.

37033        **-u**            Suppress the writing of lines that are repeated in the input.

37034 **OPERANDS**

37035        The following operands shall be supported:

37036        *input\_file*    A pathname of the input file. If the *input\_file* operand is not specified, or if the  
 37037                        *input\_file* is `'-'`, the standard input shall be used.

37038        *output\_file*    A pathname of the output file. If the *output\_file* operand is not specified, the  
 37039                        standard output shall be used. The results are unspecified if the file named by  
 37040                        *output\_file* is the file named by *input\_file*.

37041 **STDIN**

37042        The standard input shall be used only if no *input\_file* operand is specified or if *input\_file* is `'-'`.  
 37043        See the INPUT FILES section.

37044 **INPUT FILES**

37045        The input file shall be a text file.

## 37046 ENVIRONMENT VARIABLES

37047 The following environment variables shall affect the execution of *uniq*:

37048 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 37049 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 37050 Internationalization Variables for the precedence of internationalization variables  
 37051 used to determine the values of locale categories.)

37052 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 37053 internationalization variables.

37054 **LC\_COLLATE** Determine the locale for ordering rules. |  
 37055

37056 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 37057 characters (for example, single-byte as opposed to multi-byte characters in  
 37058 arguments and input files) and which characters constitute a <blank> in the  
 37059 current locale.

37060 **LC\_MESSAGES**  
 37061 Determine the locale that should be used to affect the format and contents of  
 37062 diagnostic messages written to standard error.

37063 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

## 37064 ASYNCHRONOUS EVENTS

37065 Default.

## 37066 STDOUT

37067 The standard output shall be used only if no *output\_file* operand is specified. See the OUTPUT  
 37068 FILES section.

## 37069 STDERR

37070 The standard error shall be used only for diagnostic messages.

## 37071 OUTPUT FILES

37072 If the **-c** option is specified, the output file shall be empty or each line shall be of the form: |

37073 "%d %s", <number of duplicates>, <line>

37074 otherwise, the output file shall be empty or each line shall be of the form: |

37075 "%s", <line>

## 37076 EXTENDED DESCRIPTION

37077 None.

## 37078 EXIT STATUS

37079 The following exit values shall be returned:

37080 0 The utility executed successfully.

37081 >0 An error occurred.

## 37082 CONSEQUENCES OF ERRORS

37083 Default.

37084 **APPLICATION USAGE**

37085       The *sort* utility can be used to cause repeated lines to be adjacent in the input file.

37086 **EXAMPLES**

37087       The following input file data (but flushed left) was used for a test series on *uniq*:

```
37088 #01 foo0 bar0 foo1 bar1
37089 #02 bar0 foo1 bar1 foo1
37090 #03 foo0 bar0 foo1 bar1
37091 #04
37092 #05 foo0 bar0 foo1 bar1
37093 #06 foo0 bar0 foo1 bar1
37094 #07 bar0 foo1 bar1 foo0
```

37095       What follows is a series of test invocations of the *uniq* utility that use a mixture of *uniq* options  
 37096       against the input file data. These tests verify the meaning of *adjacent*. The *uniq* utility views the  
 37097       input data as a sequence of strings delimited by '`\n`'. Accordingly, for the *fieldsth* member of  
 37098       the sequence, *uniq* interprets unique or repeated adjacent lines strictly relative to the *fields+1*th  
 37099       member.

- 37100       1. This first example tests the line counting option, comparing each line of the input file data  
 37101       starting from the second field:

```
37102 uniq -c -f 1 uniq_0I.t
37103 1 #01 foo0 bar0 foo1 bar1
37104 1 #02 bar0 foo1 bar1 foo0
37105 1 #03 foo0 bar0 foo1 bar1
37106 1 #04
37107 2 #05 foo0 bar0 foo1 bar1
37108 1 #07 bar0 foo1 bar1 foo0
```

37109       The number '2', prefixing the fifth line of output, signifies that the *uniq* utility detected a  
 37110       pair of repeated lines. Given the input data, this can only be true when *uniq* is run using  
 37111       the **-f 1** option (which shall cause *uniq* to ignore the first field on each input line).

- 37112       2. The second example tests the option to suppress unique lines, comparing each line of the  
 37113       input file data starting from the second field:

```
37114 uniq -d -f 1 uniq_0I.t
37115 #05 foo0 bar0 foo1 bar1
```

- 37116       3. This test suppresses repeated lines, comparing each line of the input file data starting from  
 37117       the second field:

```
37118 uniq -u -f 1 uniq_0I.t
37119 #01 foo0 bar0 foo1 bar1
37120 #02 bar0 foo1 bar1 foo1
37121 #03 foo0 bar0 foo1 bar1
37122 #04
37123 #07 bar0 foo1 bar1 foo0
```

- 37124       4. This suppresses unique lines, comparing each line of the input file data starting from the  
 37125       third character:

```
37126 uniq -d -s 2 uniq_0I.t
```

37127       In the last example, the *uniq* utility found no input matching the above criteria.

37128 **RATIONALE**

37129           Some historical implementations have limited lines to be 1 080 bytes in length, which does not  
37130           meet the implied {LINE\_MAX} limit.

37131 **FUTURE DIRECTIONS**

37132           None.

37133 **SEE ALSO**

37134           *comm, sort*

37135 **CHANGE HISTORY**

37136           First released in Issue 2.

37137 **Issue 6**

37138           The obsolescent SYNOPSIS and associated text are removed.

37139           The normative text is reworded to avoid use of the term “must” for application requirements.

37140           IEEE Std 1003.1-2001 / Cor 1-2002, item XCU / TC1 / D6 / 40 is applied, adding *LC\_COLLATE* to the  
37141           ENVIRONMENT VARIABLES section, and changing “the application shall ensure that” in the  
37142           OUTPUT FILES section.

37143 **NAME**37144           unlink — call the *unlink()* function37145 **SYNOPSIS**37146 xSI        unlink *file*

37147

37148 **DESCRIPTION**37149           The *unlink* utility shall perform the function call:37150           unlink(*file*);37151           A user may need appropriate privilege to invoke the *unlink* utility.37152 **OPTIONS**

37153           None.

37154 **OPERANDS**

37155           The following operands shall be supported:

37156           *file*           The pathname of an existing file.37157 **STDIN**

37158           Not used.

37159 **INPUT FILES**

37160           Not used.

37161 **ENVIRONMENT VARIABLES**37162           The following environment variables shall affect the execution of *unlink*:

37163           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 37164                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 37165           Internationalization Variables for the precedence of internationalization variables  
 37166           used to determine the values of locale categories.)

37167           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 37168           internationalization variables.

37169           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 37170           characters (for example, single-byte as opposed to multi-byte characters in  
 37171           arguments).

37172           *LC\_MESSAGES*   Determine the locale that should be used to affect the format and contents of  
 37173           diagnostic messages written to standard error.  
 37174

37175           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37176 **ASYNCHRONOUS EVENTS**

37177           Default.

37178 **STDOUT**

37179           None.

37180 **STDERR**

37181           The standard error shall be used only for diagnostic messages.

37182 **OUTPUT FILES**

37183           None.

37184 **EXTENDED DESCRIPTION**

37185           None.

37186 **EXIT STATUS**

37187           The following exit values shall be returned:

37188           0   Successful completion.

37189           >0  An error occurred.

37190 **CONSEQUENCES OF ERRORS**

37191           Default.

37192 **APPLICATION USAGE**

37193           None.

37194 **EXAMPLES**

37195           None.

37196 **RATIONALE**

37197           None.

37198 **FUTURE DIRECTIONS**

37199           None.

37200 **SEE ALSO**

37201           *link*, *rm*, the System Interfaces volume of IEEE Std 1003.1-2001, *unlink()*

37202 **CHANGE HISTORY**

37203           First released in Issue 5.



37204 **NAME**

37205           uucp — system-to-system copy

37206 **SYNOPSIS**37207 xsl       uucp [-cCdfjmr] [-n user] *source-file*... *destination-file*

37208

37209 **DESCRIPTION**37210       The *uucp* utility shall copy files named by the *source-file* argument to the *destination-file*  
37211       argument. The files named can be on local or remote systems.37212       The *uucp* utility cannot guarantee support for all character encodings in all circumstances. For  
37213       example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and  
37214       filenames need not be portable to non-internationalized systems, and so on. Under these  
37215       circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991  
37216       standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used,  
37217       and that only characters defined in the portable filename character set be used for naming files.  
37218       The protocol for transfer of files is unspecified by IEEE Std 1003.1-2001.37219       Typical implementations of this utility require a communications line configured to use the Base  
37220       Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, but other  
37221       communications means may be used. On systems where there are no available communications  
37222       means (either temporarily or permanently), this utility shall write an error message describing  
37223       the problem and exit with a non-zero exit status.37224 **OPTIONS**37225       The *uucp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
37226       12.2, Utility Syntax Guidelines.

37227       The following options shall be supported:

37228       **-c**           Do not copy local file to the spool directory for transfer to the remote machine  
37229       (default).37230       **-C**           Force the copy of local files to the spool directory for transfer.37231       **-d**           Make all necessary directories for the file copy (default).37232       **-f**           Do not make intermediate directories for the file copy.37233       **-j**           Write the job identification string to standard output. This job identification can be  
37234       used by *uustat* to obtain the status or terminate a job.37235       **-m**           Send mail to the requester when the copy is completed.37236       **-n user**       Notify *user* on the remote system that a file was sent.37237       **-r**           Do not start the file transfer; just queue the job.37238 **OPERANDS**

37239       The following operands shall be supported:

37240       *destination-file*, *source-file*37241           A pathname of a file to be copied to, or from, respectively. Either name can be a  
37242           pathname on the local machine, or can have the form:37243           *system-name*!*pathname*37244           where *system-name* is taken from a list of system names that *uucp* knows about.  
37245           The destination *system-name* can also be a list of names such as:

37246 *system-name!system-name!...!system-name!pathname*

37247 in which case, an attempt is made to send the file via the specified route to the

37248 destination. Care should be taken to ensure that intermediate nodes in the route

37249 are willing to forward information.

37250 The shell pattern matching notation characters '?', '\*', and "[...]" appearing

37251 in *pathname* shall be expanded on the appropriate system.

37252 Pathnames can be one of:

37253 1. An absolute pathname.

37254 2. A pathname preceded by *~user* where *user* is a login name on the specified

37255 system and is replaced by that user's login directory. Note that if an invalid

37256 login is specified, the default is to the public directory (called *PUBDIR*; the

37257 actual location of *PUBDIR* is implementation-defined).

37258 3. A pathname preceded by *~/destination* where *destination* is appended to

37259 *PUBDIR*.

37260 **Note:** This destination is treated as a filename unless more than one file is being

37261 transferred by this request or the destination is already a directory. To

37262 ensure that it is a directory, follow the destination with a '/'. For

37263 example, *~/dan/* as the destination makes the directory **PUBDIR/dan** if it

37264 does not exist and puts the requested files in that directory.

37265 4. Anything else shall be prefixed by the current directory.

37266 If the result is an erroneous pathname for the remote system, the copy shall fail. If

37267 the *destination-file* is a directory, the last part of the *source-file* name shall be used.

37268 The read, write, and execute permissions given by *uucp* are implementation-

37269 defined.

37270 **STDIN**

37271 Not used.

37272 **INPUT FILES**

37273 The files to be copied are regular files.

37274 **ENVIRONMENT VARIABLES**

37275 The following environment variables shall affect the execution of *uucp*:

37276 *LANG* Provide a default value for the internationalization variables that are unset or null.

37277 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,

37278 Internationalization Variables for the precedence of internationalization variables

37279 used to determine the values of locale categories.)

37280 *LC\_ALL* If set to a non-empty string value, override the values of all the other

37281 internationalization variables.

37282 *LC\_COLLATE*

37283 Determine the locale for the behavior of ranges, equivalence classes, and multi-

37284 character collating elements within bracketed filename patterns.

37285 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as

37286 characters (for example, single-byte as opposed to multi-byte characters in

37287 arguments and input files) and the behavior of character classes within bracketed

37288 filename patterns (for example, "'[:lower:])\*'").

37289 **LC\_MESSAGES**

37290 Determine the locale that should be used to affect the format and contents of  
37291 diagnostic messages written to standard error, and informative messages written  
37292 to standard output.

37293 **NLS\_PATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37294 **ASYNCHRONOUS EVENTS**

37295 Default.

37296 **STDOUT**

37297 Not used.

37298 **STDERR**

37299 The standard error shall be used only for diagnostic messages.

37300 **OUTPUT FILES**

37301 The output files (which may be on other systems) are copies of the input files.

37302 If **-m** is used, mail files are modified.

37303 **EXTENDED DESCRIPTION**

37304 None.

37305 **EXIT STATUS**

37306 The following exit values shall be returned:

37307 0 Successful completion.

37308 >0 An error occurred.

37309 **CONSEQUENCES OF ERRORS**

37310 Default.

37311 **APPLICATION USAGE**

37312 The domain of remotely accessible files can (and for obvious security reasons usually should) be  
37313 severely restricted.

37314 Note that the **'!'** character in addresses has to be escaped when using *cs***h** as a command  
37315 interpreter because of its history substitution syntax. For *ks***h** and *sh* the escape is not necessary,  
37316 but may be used.

37317 As noted above, shell metacharacters appearing in pathnames are expanded on the appropriate  
37318 system. On an internationalized system, this is done under the control of local settings of  
37319 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed filename  
37320 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
37321 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
37322 need not be supported on non-internationalized systems.

37323 **EXAMPLES**

37324 None.

37325 **RATIONALE**

37326 None.

37327 **FUTURE DIRECTIONS**

37328 None.

37329 **SEE ALSO**

37330 *mailx, uuencode, uustat, uux*

37331 **CHANGE HISTORY**

37332 First released in Issue 2.

37333 **Issue 6**

37334 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

37335 The UN margin codes and associated shading are removed from the **-C**, **-f**, **-j**, **-n**, and **-r**  
37336 options in response to The Open Group Base Resolution bwg2001-003.

37337 **NAME**

37338           uudecode — decode a binary file

37339 **SYNOPSIS**37340 UP       uudecode [-o *outfile*] [*file*]

37341

37342 **DESCRIPTION**

37343       The *uudecode* utility shall read a file, or standard input if no file is specified, that includes data  
 37344       created by the *uuencode* utility. The *uudecode* utility shall scan the input file, searching for data  
 37345       compatible with one of the formats specified in *uuencode*, and attempt to create or overwrite the  
 37346       file described by the data (or overridden by the **-o** option). The pathname shall be contained in  
 37347       the data or specified by the **-o** option. The file access permission bits and contents for the file to  
 37348       be produced shall be contained in that data. The mode bits of the created file (other than  
 37349       standard output) shall be set from the file access permission bits contained in the data; that is,  
 37350       other attributes of the mode, including the file mode creation mask (see *umask*), shall not affect  
 37351       the file being produced.

37352       If the pathname of the file to be produced exists, and the user does not have write permission on  
 37353       that file, *uudecode* shall terminate with an error. If the pathname of the file to be produced exists,  
 37354       and the user has write permission on that file, the existing file shall be overwritten.

37355       If the input data was produced by *uuencode* on a system with a different number of bits per byte  
 37356       than on the target system, the results of *uudecode* are unspecified.

37357 **OPTIONS**

37358       The *uudecode* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001,  
 37359       Section 12.2, Utility Syntax Guidelines.

37360       The following option shall be supported by the implementation:

37361       **-o *outfile***    A pathname of a file that shall be used instead of any pathname contained in the  
 37362       input data. Specifying an *outfile* option-argument of **/dev/stdout** shall indicate  
 37363       standard output.

37364 **OPERANDS**

37365       The following operand shall be supported:

37366       *file*           The pathname of a file containing the output of *uuencode*.

37367 **STDIN**

37368       See the INPUT FILES section.

37369 **INPUT FILES**

37370       The input files shall be files containing the output of *uuencode*.

37371 **ENVIRONMENT VARIABLES**

37372       The following environment variables shall affect the execution of *uudecode*:

37373       **LANG**           Provide a default value for the internationalization variables that are unset or null.  
 37374       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 37375       Internationalization Variables for the precedence of internationalization variables  
 37376       used to determine the values of locale categories.)

37377       **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 37378       internationalization variables.

37379       **LC\_CTYPE**    Determine the locale for the interpretation of sequences of bytes of text data as  
 37380       characters (for example, single-byte as opposed to multi-byte characters in  
 37381       arguments and input files).

37382 **LC\_MESSAGES**

37383 Determine the locale that should be used to affect the format and contents of  
37384 diagnostic messages written to standard error.

37385 xSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

37386 **ASYNCHRONOUS EVENTS**

37387 Default.

37388 **STDOUT**

37389 If the file data header encoded by *uencode* is **-** or **/dev/stdout**, or the **-o /dev/stdout** option  
37390 overrides the file data, the standard output shall be in the same format as the file originally  
37391 encoded by *uencode*. Otherwise, the standard output shall not be used.

37392 **STDERR**

37393 The standard error shall be used only for diagnostic messages.

37394 **OUTPUT FILES**

37395 The output file shall be in the same format as the file originally encoded by *uencode*.

37396 **EXTENDED DESCRIPTION**

37397 None.

37398 **EXIT STATUS**

37399 The following exit values shall be returned:

37400 0 Successful completion.

37401 >0 An error occurred.

37402 **CONSEQUENCES OF ERRORS**

37403 Default.

37404 **APPLICATION USAGE**

37405 The user who is invoking *uudecode* must have write permission on any file being created.

37406 The output of *uencode* is essentially an encoded bit stream that is not cognizant of byte  
37407 boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source,  
37408 if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only  
37409 data that is meaningful for such a transfer between architectures is generally character data.

37410 **EXAMPLES**

37411 None.

37412 **RATIONALE**

37413 Input files are not necessarily text files, as stated by an early proposal. Although the *uencode*  
37414 output is a text file, that output could have been wrapped within another file or mail message  
37415 that is not a text file.

37416 The **-o** option is not historical practice, but was added at the request of WG15 so that the user  
37417 could override the target pathname without having to edit the input data itself.

37418 In early drafts, the **[-o outfile]** option-argument allowed the use of **-** to mean standard output.  
37419 The symbol **-** has only been used previously in IEEE Std 1003.1-2001 as a standard input  
37420 indicator. The developers of the standard did not wish to overload the meaning of **-** in this  
37421 manner. The **/dev/stdout** concept exists on most modern systems. The **/dev/stdout** syntax does  
37422 not refer to a new special file. It is just a magic cookie to specify standard output.

37423 **FUTURE DIRECTIONS**

37424           None.

37425 **SEE ALSO**37426           *umask, uuencode*37427 **CHANGE HISTORY**

37428           First released in Issue 4.

37429 **Issue 6**

37430           This utility is marked as part of the User Portability Utilities option.

37431           The **-o** *outfile* option is added, as specified in the IEEE P1003.2b draft standard.

37432           The normative text is reworded to avoid use of the term “must” for application requirements.

37433 **NAME**

37434 uuencode — encode a binary file

37435 **SYNOPSIS**37436 UP uuencode [-m] [*file*] *decode\_pathname*

37437

37438 **DESCRIPTION**

37439 The *uuencode* utility shall write an encoded version of the named input file, or standard input if  
 37440 no *file* is specified, to standard output. The output shall be encoded using one of the algorithms  
 37441 described in the STDOUT section and shall include the file access permission bits (in *chmod* octal  
 37442 or symbolic notation) of the input file and the *decode\_pathname*, for re-creation of the file on  
 37443 another system that conforms to this volume of IEEE Std 1003.1-2001.

37444 **OPTIONS**

37445 The *uuencode* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001,  
 37446 Section 12.2, Utility Syntax Guidelines.

37447 The following option shall be supported by the implementation:

37448 **-m** Encode the output using the MIME Base64 algorithm described in STDOUT. If **-m**  
 37449 is not specified, the historical algorithm described in STDOUT shall be used.

37450 **OPERANDS**

37451 The following operands shall be supported:

37452 *decode\_pathname*

37453 The pathname of the file into which the *uudecode* utility shall place the decoded  
 37454 file. Specifying a *decode\_pathname* operand of **/dev/stdout** shall indicate that  
 37455 *uudecode* is to use standard output. If there are characters in *decode\_pathname* that  
 37456 are not in the portable filename character set the results are unspecified.

37457 *file* A pathname of the file to be encoded.

37458 **STDIN**

37459 See the INPUT FILES section.

37460 **INPUT FILES**

37461 Input files can be files of any type.

37462 **ENVIRONMENT VARIABLES**

37463 The following environment variables shall affect the execution of *uuencode*:

37464 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 37465 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 37466 Internationalization Variables for the precedence of internationalization variables  
 37467 used to determine the values of locale categories.)

37468 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 37469 internationalization variables.

37470 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 37471 characters (for example, single-byte as opposed to multi-byte characters in  
 37472 arguments and input files).

37473 **LC\_MESSAGES**

37474 Determine the locale that should be used to affect the format and contents of  
 37475 diagnostic messages written to standard error.



37476 xSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 37477 ASYNCHRONOUS EVENTS

37478 Default.

## 37479 STDOUT

### 37480 uuencode Base64 Algorithm

37481 The standard output shall be a text file (encoded in the character set of the current locale) that  
37482 begins with the line:

37483 "begin-base64 $\Delta$ %s $\Delta$ %s\n", <mode>, <decode\_pathname>

37484 and ends with the line:

37485 "====\n"

37486 In both cases, the lines shall have no preceding or trailing <blank>s.

37487 The encoding process represents 24-bit groups of input bits as output strings of four encoded  
37488 characters. Proceeding from left to right, a 24-bit input group shall be formed by concatenating  
37489 three 8-bit input groups. Each 24-bit input group then shall be treated as four concatenated 6-bit  
37490 groups, each of which shall be translated into a single digit in the Base64 alphabet. When  
37491 encoding a bit stream via the Base64 encoding, the bit stream shall be presumed to be ordered  
37492 with the most-significant bit first. That is, the first bit in the stream shall be the high-order bit in  
37493 the first byte, and the eighth bit shall be the low-order bit in the first byte, and so on. Each 6-bit  
37494 group is used as an index into an array of 64 printable characters, as shown in Table 4-21.

37495 **Table 4-21** uuencode Base64 Values

| Value | Encoding | Value | Encoding | Value | Encoding | Value | Encoding |
|-------|----------|-------|----------|-------|----------|-------|----------|
| 0     | A        | 17    | R        | 34    | i        | 51    | z        |
| 1     | B        | 18    | S        | 35    | j        | 52    | 0        |
| 2     | C        | 19    | T        | 36    | k        | 53    | 1        |
| 3     | D        | 20    | U        | 37    | l        | 54    | 2        |
| 4     | E        | 21    | V        | 38    | m        | 55    | 3        |
| 5     | F        | 22    | W        | 39    | n        | 56    | 4        |
| 6     | G        | 23    | X        | 40    | o        | 57    | 5        |
| 7     | H        | 24    | Y        | 41    | p        | 58    | 6        |
| 8     | I        | 25    | Z        | 42    | q        | 59    | 7        |
| 9     | J        | 26    | a        | 43    | r        | 60    | 8        |
| 10    | K        | 27    | b        | 44    | s        | 61    | 9        |
| 11    | L        | 28    | c        | 45    | t        | 62    | +        |
| 12    | M        | 29    | d        | 46    | u        | 63    | /        |
| 13    | N        | 30    | e        | 47    | v        | (pad) | =        |
| 14    | O        | 31    | f        | 48    | w        |       |          |
| 15    | P        | 32    | g        | 49    | x        |       |          |
| 16    | Q        | 33    | h        | 50    | y        |       |          |

37514 The character referenced by the index shall be placed in the output string.

37515 The output stream (encoded bytes) shall be represented in lines of no more than 76 characters  
37516 each. All line breaks or other characters not found in the table shall be ignored by decoding  
37517 software (see *uudecode*).

37518 Special processing shall be performed if fewer than 24 bits are available at the end of a message  
37519 or encapsulated part of a message. A full encoding quantum shall always be completed at the

37520 end of a message. When fewer than 24 input bits are available in an input group, zero bits shall  
 37521 be added (on the right) to form an integral number of 6-bit groups. Output character positions  
 37522 that are not required to represent actual input data shall be set to the character '='. Since all  
 37523 Base64 input is an integral number of octets, only the following cases can arise:

- 37524 1. The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of  
 37525 encoded output shall be an integral multiple of 4 characters with no '=' padding.
- 37526 2. The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded  
 37527 output shall be three characters followed by one '=' padding character.
- 37528 3. The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output  
 37529 shall be two characters followed by two '=' padding characters.

37530 A terminating "====" evaluates to nothing and denotes the end of the encoded data.

### 37531 uuencode Historical Algorithm

37532 The standard output shall be a text file (encoded in the character set of the current locale) that  
 37533 begins with the line:

37534 "beginΔsΔs\n" <mode>, <decode\_pathname>

37535 and ends with the line:

37536 "end\n"

37537 In both cases, the lines shall have no preceding or trailing <blank>s.

37538 The algorithm that shall be used for lines in between **begin** and **end** takes three octets as input  
 37539 and writes four characters of output by splitting the input at six-bit intervals into four octets,  
 37540 containing data in the lower six bits only. These octets shall be converted to characters by adding  
 37541 a value of 0x20 to each octet, so that each octet is in the range [0x20,0x5f], and then it shall be  
 37542 assumed to represent a printable character in the ISO/IEC 646:1991 standard encoded character  
 37543 set. It then shall be translated into the corresponding character codes for the codeset in use in the  
 37544 current locale. (For example, the octet 0x41, representing 'A', would be translated to 'A' in the  
 37545 current codeset, such as 0xc1 if it were EBCDIC.)

37546 Where the bits of two octets are combined, the least significant bits of the first octet shall be  
 37547 shifted left and combined with the most significant bits of the second octet shifted right. Thus  
 37548 the three octets A, B, C shall be converted into the four octets:

```
37549 0x20 + ((A >> 2) & 0x3F)
37550 0x20 + (((A << 4) | ((B >> 4) & 0xF)) & 0x3F)
37551 0x20 + (((B << 2) | ((C >> 6) & 0x3)) & 0x3F)
37552 0x20 + ((C) & 0x3F)
```

37553 These octets then shall be translated into the local character set.

37554 Each encoded line contains a length character, equal to the number of characters to be decoded  
 37555 plus 0x20 translated to the local character set as described above, followed by the encoded  
 37556 characters. The maximum number of octets to be encoded on each line shall be 45.

### 37557 STDERR

37558 The standard error shall be used only for diagnostic messages.

### 37559 OUTPUT FILES

37560 None.

37561 **EXTENDED DESCRIPTION**

37562       None.

37563 **EXIT STATUS**

37564       The following exit values shall be returned:

37565       0   Successful completion.

37566       &gt;0   An error occurred.

37567 **CONSEQUENCES OF ERRORS**

37568       Default.

37569 **APPLICATION USAGE**

37570       The file is expanded by 35 percent (each three octets become four, plus control information)  
37571       causing it to take longer to transmit.

37572       Since this utility is intended to create files to be used for data interchange between systems with  
37573       possibly different codesets, and to represent binary data as a text file, the ISO/IEC 646:1991  
37574       standard was chosen for a midpoint in the algorithm as a known reference point. The output  
37575       from *uuencode* is a text file on the local system. If the output were in the ISO/IEC 646:1991  
37576       standard codeset, it might not be a text file (at least because the <newline>s might not match),  
37577       and the goal of creating a text file would be defeated. If this text file was then carried to another  
37578       machine with the same codeset, it would be perfectly compatible with that system's *uudecode*. If  
37579       it was transmitted over a mail system or sent to a machine with a different codeset, it is assumed  
37580       that, as for every other text file, some translation mechanism would convert it (by the time it  
37581       reached a user on the other system) into an appropriate codeset. This translation only makes  
37582       sense from the local codeset, not if the file has been put into a ISO/IEC 646:1991 standard  
37583       representation first. Similarly, files processed by *uuencode* can be placed in *pax* archives,  
37584       intermixed with other text files in the same codeset.

37585 **EXAMPLES**

37586       None.

37587 **RATIONALE**

37588       A new algorithm was added at the request of the international community to parallel work in  
37589       RFC 2045 (MIME). As with the historical *uuencode* format, the Base64 Content-Transfer-Encoding  
37590       is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A  
37591       65-character subset of the ISO/IEC 646:1991 standard is used, enabling 6 bits to be represented  
37592       per printable character. (The extra 65th character, '=', is used to signify a special processing  
37593       function.)

37594       This subset has the important property that it is represented identically in all versions of the  
37595       ISO/IEC 646:1991 standard, including US ASCII, and all characters in the subset are also  
37596       represented identically in all versions of EBCDIC. The historical *uuencode* algorithm does not  
37597       share this property, which is the reason that a second algorithm was added to the ISO POSIX-2  
37598       standard.

37599       The string "====" was used for the termination instead of the end used in the original format  
37600       because the latter is a string that could be valid encoded input.

37601       In an early draft, the **-m** option was named **-b** (for Base64), but it was renamed to reflect its  
37602       relationship to the RFC 2045. A **-u** was also present to invoke the default algorithm, but since  
37603       this was not historical practice, it was omitted as being unnecessary.

37604       See the RATIONALE section in *uudecode* for the derivation of the **/dev/stdout** symbol.

37605 **FUTURE DIRECTIONS**

37606           None.

37607 **SEE ALSO**37608           *chmod, mailx, uudecode*37609 **CHANGE HISTORY**

37610           First released in Issue 4.

37611 **Issue 6**

37612           This utility is marked as part of the User Portability Utilities option.

37613           The Base64 algorithm and the ability to output to **/dev/stdout** are added as specified in the  
37614           IEEE P1003.2b draft standard.

37615 **NAME**

37616           uustat — uucp status inquiry and job control

37617 **SYNOPSIS**37618 xSI       uustat [ -q | -k *jobid* | -r *jobid* ]37619           uustat [-s *system*] [-u *user*]

37620

37621 **DESCRIPTION**37622           The *uustat* utility shall display the status of, or cancel, previously specified *uucp* requests, or  
37623           provide general status on *uucp* connections to other systems.37624           When no options are given, *uustat* shall write to standard output the status of all *uucp* requests  
37625           issued by the current user.37626           Typical implementations of this utility require a communications line configured to use the Base  
37627           Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, but other  
37628           communications means may be used. On systems where there are no available communications  
37629           means (either temporarily or permanently), this utility shall write an error message describing  
37630           the problem and exit with a non-zero exit status.37631 **OPTIONS**37632           The *uustat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
37633           12.2, Utility Syntax Guidelines.

37634           The following options shall be supported:

37635           **-q**           Write the jobs queued for each machine.37636           **-k** *jobid*       Kill the *uucp* request whose job identification is *jobid*. The application shall ensure  
37637           that the killed *uucp* request belongs to the person invoking *uustat* unless that user  
37638           has appropriate privileges.37639           **-r** *jobid*       Rejuvenate *jobid*. The files associated with *jobid* are touched so that their  
37640           modification time is set to the current time. This prevents the cleanup program  
37641           from deleting the job until the jobs modification time reaches the limit imposed by  
37642           the program.37643           **-s** *system*       Write the status of all *uucp* requests for remote system *system*.37644           **-u** *user*       Write the status of all *uucp* requests issued by *user*.37645 **OPERANDS**

37646           None.

37647 **STDIN**

37648           Not used.

37649 **INPUT FILES**

37650           None.

37651 **ENVIRONMENT VARIABLES**37652           The following environment variables shall affect the execution of *uustat*:37653           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
37654           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
37655           Internationalization Variables for the precedence of internationalization variables  
37656           used to determine the values of locale categories.)

37657 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
37658 internationalization variables.

37659 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
37660 characters (for example, single-byte as opposed to multi-byte characters in  
37661 arguments).

37662 *LC\_MESSAGES*  
37663 Determine the locale that should be used to affect the format and contents of  
37664 diagnostic messages written to standard error, and informative messages written  
37665 to standard output.

37666 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37667 **ASYNCHRONOUS EVENTS**  
37668 Default.

37669 **STDOUT**  
37670 The standard output shall consist of information about each job selected, in an unspecified  
37671 format. The information shall include at least the job ID, the user ID or name, and the remote  
37672 system name.

37673 **STDERR**  
37674 The standard error shall be used only for diagnostic messages.

37675 **OUTPUT FILES**  
37676 None.

37677 **EXTENDED DESCRIPTION**  
37678 None.

37679 **EXIT STATUS**  
37680 The following exit values shall be returned:  
37681 0 Successful completion.  
37682 >0 An error occurred.

37683 **CONSEQUENCES OF ERRORS**  
37684 Default.

37685 **APPLICATION USAGE**  
37686 None.

37687 **EXAMPLES**  
37688 None.

37689 **RATIONALE**  
37690 None.

37691 **FUTURE DIRECTIONS**  
37692 None.

37693 **SEE ALSO**  
37694 *uucp*

37695 **CHANGE HISTORY**  
37696 First released in Issue 2.

37697 **Issue 6**

37698 The normative text is reworded to avoid use of the term “must” for application requirements.

37699 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

37700 The UN margin code and associated shading are removed from the **-q** option in response to The  
37701 Open Group Base Resolution bwg2001-003.

37702 **NAME**

37703       uux — remote command execution

37704 **SYNOPSIS**37705 xsi       uux [-np] *command-string*37706       uux [-jnp] *command-string*

37707

37708 **DESCRIPTION**

37709       The *uux* utility shall gather zero or more files from various systems, execute a shell pipeline (see  
 37710       Section 2.9 (on page 47)) on a specified system, and then send the standard output of the  
 37711       command to a file on a specified system. Only the first command of a pipeline can have a  
 37712       *system-name!* prefix. All other commands in the pipeline shall be executed on the system of the  
 37713       first command.

37714       The following restrictions are applicable to the shell pipeline processed by *uux*:

- 37715       • In gathering files from different systems, pathname expansion shall not be performed by *uux*.  
 37716       Thus, a request such as:

37717       uux "c99 remsys!~/\*.c"

37718       would attempt to copy the file named literally *\*.c* to the local system.

- 37719       • The redirection operators ">>", "<<", ">|", and ">&" shall not be accepted. Any use of  
 37720       these redirection operators shall cause this utility to write an error message describing the  
 37721       problem and exit with a non-zero exit status.

- 37722       • The reserved word *!* cannot be used at the head of the pipeline to modify the exit status. (See  
 37723       the *command-string* operand description below.)

- 37724       • Alias substitution shall not be performed.

37725       A filename can be specified as for *uucp*; it can be an absolute pathname, a pathname preceded by  
 37726       ~*name* (which is replaced by the corresponding login directory), a pathname specified as ~/ *dest*  
 37727       (*dest* is prefixed by the public directory called *PUBDIR*; the actual location of *PUBDIR* is  
 37728       implementation-defined), or a simple filename (which is prefixed by *uux* with the current  
 37729       directory). See *uucp* for the details.

37730       The execution of commands on remote systems shall take place in an execution directory known  
 37731       to the *uucp* system. All files required for the execution shall be put into this directory unless they  
 37732       already reside on that machine. Therefore, the application shall ensure that non-local filenames  
 37733       (without path or machine reference) are unique within the *uux* request.

37734       The *uux* utility shall attempt to get all files to the execution system. For files that are output files,  
 37735       the application shall ensure that the filename is escaped using parentheses.

37736       The remote system shall notify the user by mail if the requested command on the remote system  
 37737       was disallowed or the files were not accessible. This notification can be turned off by the **-n**  
 37738       option.

37739       Typical implementations of this utility require a communications line configured to use the Base  
 37740       Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, but other  
 37741       communications means may be used. On systems where there are no available communications  
 37742       means (either temporarily or permanently), this utility shall write an error message describing  
 37743       the problem and exit with a non-zero exit status.

37744       The *uux* utility cannot guarantee support for all character encodings in all circumstances. For  
 37745       example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and



37746 filenames need not be portable to non-internationalized systems, and so on. Under these  
 37747 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991  
 37748 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used  
 37749 and that only characters defined in the portable filename character set be used for naming files.

### 37750 OPTIONS

37751 The *uux* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 37752 12.2, Utility Syntax Guidelines.

37753 The following options shall be supported:

- 37754 **-p** Make the standard input to *uux* the standard input to the *command-string*.
- 37755 **-j** Write the job identification string to standard output. This job identification can be  
 37756 used by *uustat* to obtain the status or terminate a job.
- 37757 **-n** Do not notify the user if the command fails.

### 37758 OPERANDS

37759 The following operand shall be supported:

- 37760 *command-string*  
 37761 A string made up of one or more arguments that are similar to normal command  
 37762 arguments, except that the command and any filenames can be prefixed by  
 37763 *system-name*!. A null *system-name* shall be interpreted as the local system.

### 37764 STDIN

37765 The standard input shall not be used unless the **'-'** or **-p** option is specified; in those cases, the  
 37766 standard input shall be made the standard input of the *command-string*.

### 37767 INPUT FILES

37768 Input files shall be selected according to the contents of *command-string*.

### 37769 ENVIRONMENT VARIABLES

37770 The following environment variables shall affect the execution of *uux*:

- 37771 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 37772 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 37773 Internationalization Variables for the precedence of internationalization variables  
 37774 used to determine the values of locale categories.)
- 37775 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 37776 internationalization variables.
- 37777 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 37778 characters (for example, single-byte as opposed to multi-byte characters in  
 37779 arguments).
- 37780 **LC\_MESSAGES**  
 37781 Determine the locale that should be used to affect the format and contents of  
 37782 diagnostic messages written to standard error.
- 37783 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

### 37784 ASYNCHRONOUS EVENTS

37785 Default.

37786 **STDOUT**

37787 The standard output shall not be used unless the **-j** option is specified; in that case, the job  
 37788 identification string shall be written to standard output in the following format:

37789 "%s\n", <*jobid*>

37790 **STDERR**

37791 The standard error shall be used only for diagnostic messages.

37792 **OUTPUT FILES**

37793 Output files shall be created or written, or both, according to the contents of *command-string*.

37794 If **-n** is not used, mail files shall be modified following any command or file-access failures on  
 37795 the remote system.

37796 **EXTENDED DESCRIPTION**

37797 None.

37798 **EXIT STATUS**

37799 The following exit values shall be returned:

37800 0 Successful completion.

37801 >0 An error occurred.

37802 **CONSEQUENCES OF ERRORS**

37803 Default.

37804 **APPLICATION USAGE**

37805 Note that, for security reasons, many installations limit the list of commands executable on  
 37806 behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail  
 37807 via *uux*.

37808 Any characters special to the command interpreter should be quoted either by quoting the entire  
 37809 *command-string* or quoting the special characters as individual arguments.

37810 As noted in *uucp*, shell pattern matching notation characters appearing in pathnames are  
 37811 expanded on the appropriate local system. This is done under the control of local settings of  
 37812 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed filename  
 37813 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
 37814 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
 37815 need not be supported on non-internationalized systems.

37816 **EXAMPLES**

37817 1. The following command gets **file1** from system **a** and **file2** from system **b**, executes *diff* on  
 37818 the local system, and puts the results in **file.diff** in the local *PUBDIR* directory. (*PUBDIR* is  
 37819 the *uucp* public directory on the local system.)

37820 `uux "!diff a!/usr/file1 b!/a4/file2 >!/~/file.diff"`

37821 2. The following command fails because *uux* places all files copied to a system in the same  
 37822 working directory. Although the files **xyz** are from two different systems, their filenames  
 37823 are the same and conflict.

37824 `uux "!diff a!/usr1/xyz b!/usr2/xyz >!/~/xyz.diff"`

37825 3. The following command succeeds (assuming *diff* is permitted on system **a**) because the file  
 37826 local to system **a** is not copied to the working directory, and hence does not conflict with  
 37827 the file from system **c**.

37828                   uux "a!diff a!/usr/xyz c!/usr/xyz >!~/xyz.diff"

37829 **RATIONALE**

37830                   None.

37831 **FUTURE DIRECTIONS**

37832                   None.

37833 **SEE ALSO**

37834                   Chapter 2 (on page 29), *uucp*, *uuencode*, *uustat*

37835 **CHANGE HISTORY**

37836                   First released in Issue 2.

37837 **Issue 6**

37838                   The obsolescent SYNOPSIS is removed.

37839                   The normative text is reworded to avoid use of the term “must” for application requirements.

37840                   The UN margin code and associated shading are removed from the **-j** option in response to The  
37841                   Open Group Base Resolution bwg2001-003.

37842 **NAME**37843       val — validate SCCS files (**DEVELOPMENT**)37844 **SYNOPSIS**

37845 xSI       val -

37846       val [-s] [-m *name*] [-r *SID*] [-y *type*] *file*...

37847

37848 **DESCRIPTION**37849       The *val* utility shall determine whether the specified *file* is an SCCS file meeting the  
37850       characteristics specified by the options.37851 **OPTIONS**37852       The *val* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
37853       Utility Syntax Guidelines, except that the usage of the '-' operand is not strictly as intended by  
37854       the guidelines (that is, reading options and operands from standard input).

37855       The following options shall be supported:

37856       -m *name*       Specify a *name*, which is compared with the SCCS %M% keyword in *file*; see *get*.37857       -r *SID*       Specify a *SID* (SCCS Identification String), an SCCS delta number. A check shall be  
37858       made to determine whether the *SID* is ambiguous (for example, -r 1 is ambiguous  
37859       because it physically does not exist but implies 1.1, 1.2, and so on, which may  
37860       exist) or invalid (for example, -r 1.0 or -r 1.1.0 are invalid because neither case can  
37861       exist as a valid delta number). If the *SID* is valid and not ambiguous, a check shall  
37862       be made to determine whether it actually exists.37863       -s           Silence the diagnostic message normally written to standard output for any error  
37864       that is detected while processing each named file on a given command line.37865       -y *type*       Specify a *type*, which shall be compared with the SCCS %Y% keyword in *file*; see  
37866       *get*.37867 **OPERANDS**

37868       The following operands shall be supported:

37869       *file*       A pathname of an existing SCCS file. If exactly one *file* operand appears, and it is  
37870       '-', the standard input shall be read: each line shall be independently processed  
37871       as if it were a command line argument list. (However, the line is not subjected to  
37872       any of the shell word expansions, such as parameter expansion or quote removal.)37873 **STDIN**37874       The standard input shall be a text file used only when the *file* operand is specified as '-'.37875 **INPUT FILES**

37876       Any SCCS files processed shall be files of an unspecified format.

37877 **ENVIRONMENT VARIABLES**37878       The following environment variables shall affect the execution of *val*:37879       LANG       Provide a default value for the internationalization variables that are unset or null.  
37880       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
37881       Internationalization Variables for the precedence of internationalization variables  
37882       used to determine the values of locale categories.)37883       LC\_ALL       If set to a non-empty string value, override the values of all the other  
37884       internationalization variables.

37885        **LC\_CTYPE**    Determine the locale for the interpretation of sequences of bytes of text data as  
 37886                   characters (for example, single-byte as opposed to multi-byte characters in  
 37887                   arguments and input files).

37888        **LC\_MESSAGES**  
 37889                   Determine the locale that should be used to affect the format and contents of  
 37890                   diagnostic messages written to standard error, and informative messages written  
 37891                   to standard output.

37892        **NLSPATH**    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37893 **ASYNCHRONOUS EVENTS**  
 37894        Default.

37895 **STDOUT**  
 37896        The standard output shall consist of informative messages about either:

37897        1. Each file processed

37898        2. Each command line read from standard input

37899        If the standard input is not used, for each *file* operand yielding a discrepancy, the output line  
 37900        shall have the following format:

37901        "%s: %s\n", <pathname>, <unspecified string>

37902        If standard input is used, a line of input shall be written before each of the preceding lines for  
 37903        files containing discrepancies:

37904        "%s:\n", <input line>

37905 **STDERR**  
 37906        Not used.

37907 **OUTPUT FILES**  
 37908        None.

37909 **EXTENDED DESCRIPTION**  
 37910        None.

37911 **EXIT STATUS**  
 37912        The 8-bit code returned by *val* shall be a disjunction of the possible errors; that is, it can be  
 37913        interpreted as a bit string where set bits are interpreted as follows:

37914        0x80    =    Missing file argument.

37915        0x40    =    Unknown or duplicate option.

37916        0x20    =    Corrupted SCCS file.

37917        0x10    =    Cannot open file or file not SCCS.

37918        0x08    =    *SID* is invalid or ambiguous.

37919        0x04    =    *SID* does not exist.

37920        0x02    =    %Y%, -y mismatch.

37921        0x01    =    %M%, -m mismatch.

37922        Note that *val* can process two or more files on a given command line and can process multiple  
 37923        command lines (when reading the standard input). In these cases an aggregate code shall be  
 37924        returned: a logical OR of the codes generated for each command line and file processed.

37925 **CONSEQUENCES OF ERRORS**

37926 Default.

37927 **APPLICATION USAGE**

37928 Since the *val* exit status sets the 0x80 bit, shell applications checking "\$?" cannot tell if it  
37929 terminated due to a missing file argument or receipt of a signal.

37930 **EXAMPLES**

37931 In a directory with three SCCS files—**s.x** (of **t** type “text”), **s.y**, and **s.z** (a corrupted file)—the  
37932 following command could produce the output shown:

37933 `val - <<EOF`37934 `-y source s.x`37935 `-m y s.y`37936 `s.z`37937 `EOF`37938 `-y source s.x`37939 `s.x: %Y%, -y mismatch`37940 `s.z`37941 `s.z: corrupted SCCS file`37942 **RATIONALE**

37943 None.

37944 **FUTURE DIRECTIONS**

37945 None.

37946 **SEE ALSO**37947 *admin, delta, get, prs*37948 **CHANGE HISTORY**

37949 First released in Issue 2.

37950 **Issue 6**

37951 The Open Group Corrigendum U025/4 is applied, correcting a typographical error in the EXIT  
37952 STATUS.

37953 **NAME**37954 `vi` — screen-oriented (visual) display editor37955 **SYNOPSIS**37956 UP `vi [-rR] [-c command] [-t tagstring] [-w size] [file ...]`

37957

37958 **DESCRIPTION**37959 This utility shall be provided on systems that both support the User Portability Utilities option  
37960 and define the POSIX2\_CHAR\_TERM symbol. On other systems it is optional.37961 The *vi* (visual) utility is a screen-oriented text editor. Only the open and visual modes of the  
37962 editor are described in IEEE Std 1003.1-2001; see the line editor *ex* for additional editing  
37963 capabilities used in *vi*. The user can switch back and forth between *vi* and *ex* and execute *ex*  
37964 commands from within *vi*.37965 This reference page uses the term *edit buffer* to describe the current working text. No specific  
37966 implementation is implied by this term. All editing changes are performed on the edit buffer,  
37967 and no changes to it shall affect any file until an editor command writes the file.37968 When using *vi*, the terminal screen acts as a window into the editing buffer. Changes made to  
37969 the editing buffer shall be reflected in the screen display; the position of the cursor on the screen  
37970 shall indicate the position within the editing buffer.37971 Certain terminals do not have all the capabilities necessary to support the complete *vi* definition.  
37972 When these commands cannot be supported on such terminals, this condition shall not produce  
37973 an error message such as “not an editor command” or report a syntax error. The implementation  
37974 may either accept the commands and produce results on the screen that are the result of an  
37975 unsuccessful attempt to meet the requirements of this volume of IEEE Std 1003.1-2001 or report  
37976 an error describing the terminal-related deficiency.37977 **OPTIONS**37978 The *vi* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
37979 Utility Syntax Guidelines.

37980 The following options shall be supported:

37981 `-c command` See the *ex* command description of the `-c` option.37982 `-r` See the *ex* command description of the `-r` option.37983 `-R` See the *ex* command description of the `-R` option.37984 `-t tagstring` See the *ex* command description of the `-t` option.37985 `-w size` See the *ex* command description of the `-w` option.37986 **OPERANDS**37987 See the OPERANDS section of the *ex* command for a description of the operands supported by  
37988 the *vi* command.37989 **STDIN**37990 If standard input is not a terminal device, the results are undefined. The standard input consists  
37991 of a series of commands and input text, as described in the EXTENDED DESCRIPTION section.37992 If a read from the standard input returns an error, or if the editor detects an end-of-file condition  
37993 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

37994 **INPUT FILES**

37995 See the INPUT FILES section of the *ex* command for a description of the input files supported by  
 37996 the *vi* command.

37997 **ENVIRONMENT VARIABLES**

37998 See the ENVIRONMENT VARIABLES section of the *ex* command for the environment variables  
 37999 that affect the execution of the *vi* command.

38000 **ASYNCHRONOUS EVENTS**

38001 See the ASYNCHRONOUS EVENTS section of the *ex* for the asynchronous events that affect the  
 38002 execution of the *vi* command.

38003 **STDOUT**

38004 If standard output is not a terminal device, undefined results occur.

38005 Standard output may be used for writing prompts to the user, for informational messages, and  
 38006 for writing lines from the file.

38007 **STDERR**

38008 If standard output is not a terminal device, undefined results occur.

38009 The standard error shall be used only for diagnostic messages.

38010 **OUTPUT FILES**

38011 See the OUTPUT FILES section of the *ex* command for a description of the output files  
 38012 supported by the *vi* command.

38013 **EXTENDED DESCRIPTION**

38014 If the terminal does not have the capabilities necessary to support an unspecified portion of the  
 38015 *vi* definition, implementations shall start initially in *ex* mode or open mode. Otherwise, after  
 38016 initialization, *vi* shall be in command mode; text input mode can be entered by one of several  
 38017 commands used to insert or change text. In text input mode, <ESC> can be used to return to  
 38018 command mode; other uses of <ESC> are described later in this section; see **Terminate**  
 38019 **Command or Input Mode** (on page 996).

38020 **Initialization in *ex* and *vi***

38021 See **Initialization in *ex* and *vi*** (on page 358) for a description of *ex* and *vi* initialization for the *vi*  
 38022 utility.

38023 **Command Descriptions in *vi***

38024 The following symbols are used in this reference page to represent arguments to commands.

38025 *buffer* See the description of *buffer* in the EXTENDED DESCRIPTION section of the *ex* utility;  
 38026 see **Command Descriptions in *ex*** (on page 368).

38027 In open and visual mode, when a command synopsis shows both [*buffer*] and [*count*]  
 38028 preceding the command name, they can be specified in either order.

38029 *count* A positive integer used as an optional argument to most commands, either to give a  
 38030 repeat count or as a size. This argument is optional and shall default to 1 unless  
 38031 otherwise specified.

38032 The Synopsis lines for the *vi* commands <control>-G, <control>-L, <control>-R,  
 38033 <control>-], %, &, ^, **D**, **m**, **M**, **Q**, **u**, **U**, and **ZZ** do not have *count* as an optional  
 38034 argument. Regardless, it shall not be an error to specify a *count* to these commands, and  
 38035 any specified *count* shall be ignored.



38036 *motion* An optional trailing argument used by the **!**, **<**, **>**, **c**, **d**, and **y** commands, which is used  
 38037 to indicate the region of text that shall be affected by the command. The motion can be  
 38038 either one of the command characters repeated or one of several other *vi* commands  
 38039 (listed in the following table). Each of the applicable commands specifies the region of  
 38040 text matched by repeating the command; each command that can be used as a motion  
 38041 command specifies the region of text it affects.

38042 Commands that take *motion* arguments operate on either lines or characters, depending  
 38043 on the circumstances. When operating on lines, all lines that fall partially or wholly  
 38044 within the text region specified for the command shall be affected. When operating on  
 38045 characters, only the exact characters in the specified text region shall be affected. Each  
 38046 motion command specifies this individually.

38047 When commands that may be motion commands are not used as motion commands,  
 38048 they shall set the current position to the current line and column as specified.

38049 The following commands shall be valid cursor motion commands:

|       |                   |    |    |   |   |
|-------|-------------------|----|----|---|---|
| 38050 | <apostrophe>      | (  | -  | j | H |
| 38051 | <carriage-return> | )  | \$ | k | L |
| 38052 | <comma>           | [  | %  | l | M |
| 38053 | <control>-H       | ]] | _  | n | N |
| 38054 | <control>-N       | {  | ;  | t | T |
| 38055 | <control>-P       | }  | ?  | w | W |
| 38056 | <grave accent>    | ^  | b  | B |   |
| 38057 | <newline>         | +  | e  | E |   |
| 38058 | <space>           |    | f  | F |   |
| 38059 | <zero>            | /  | h  | G |   |

38060 Any *count* that is specified to a command that has an associated motion command shall  
 38061 be applied to the motion command. If a *count* is applied to both the command and its  
 38062 associated motion command, the effect shall be multiplicative.

38063 The following symbols are used in this section to specify locations in the edit buffer:

38064 *current character*

38065 The character that is currently indicated by the cursor.

38066 *end of a line*

38067 The point located between the last non-<newline> (if any) and the terminating  
 38068 <newline> of a line. For an empty line, this location coincides with the beginning of the  
 38069 line.

38070 *end of the edit buffer*

38071 The location corresponding to the end of the last line in the edit buffer.

38072 The following symbols are used in this section to specify command actions:

38073 *bigword* In the POSIX locale, *vi* shall recognize four kinds of *bigwords*:

- 38074 1. A maximal sequence of non-<blank>s preceded and followed by <blank>s or the  
 38075 beginning or end of a line or the edit buffer
- 38076 2. One or more sequential blank lines
- 38077 3. The first character in the edit buffer
- 38078 4. The last non-<newline> in the edit buffer

- 38079        *word*        In the POSIX locale, *vi* shall recognize five kinds of words:
- 38080                1. A maximal sequence of letters, digits, and underscores, delimited at both ends by:
- 38081                    — Characters other than letters, digits, or underscores
- 38082                    — The beginning or end of a line
- 38083                    — The beginning or end of the edit buffer
- 38084                2. A maximal sequence of characters other than letters, digits, underscores, or
- 38085                    <blank>s, delimited at both ends by:
- 38086                    — A letter, digit, underscore
- 38087                    — <blank>s
- 38088                    — The beginning or end of a line
- 38089                    — The beginning or end of the edit buffer
- 38090                3. One or more sequential blank lines
- 38091                4. The first character in the edit buffer
- 38092                5. The last non-<newline> in the edit buffer
- 38093        *section boundary*
- 38094                A *section boundary* is one of the following:
- 38095                    1. A line whose first character is a <form-feed>
- 38096                    2. A line whose first character is an open curly brace ( ' { ' )
- 38097                    3. A line whose first character is a period and whose second and third characters
- 38098                    match a two-character pair in the **sections** edit option (see *ed*)
- 38099                    4. A line whose first character is a period and whose only other character matches
- 38100                    the first character of a two-character pair in the **sections** edit option, where the
- 38101                    second character of the two-character pair is a <space>
- 38102                    5. The first line of the edit buffer
- 38103                    6. The last line of the edit buffer if the last line of the edit buffer is empty or if it is a
- 38104                    **]]** or **}** command; otherwise, the last non-<newline> of the last line of the edit
- 38105                    buffer
- 38106        *paragraph boundary*
- 38107                A *paragraph boundary* is one of the following:
- 38108                    1. A section boundary
- 38109                    2. A line whose first character is a period and whose second and third characters
- 38110                    match a two-character pair in the **paragraphs** edit option (see *ed*)
- 38111                    3. A line whose first character is a period and whose only other character matches
- 38112                    the first character of a two-character pair in the *paragraphs* edit option, where the
- 38113                    second character of the two-character pair is a <space>
- 38114                    4. One or more sequential blank lines
- 38115        *remembered search direction*
- 38116                See the description of *remembered search direction* in *ed*.

38117 *sentence boundary*

38118 A *sentence boundary* is one of the following:

- 38119 1. A paragraph boundary
- 38120 2. The first non-`<blank>` that occurs after a paragraph boundary
- 38121 3. The first non-`<blank>` that occurs after a period (`' . '`), exclamation mark (`' ! '`),  
38122 or question mark (`' ? '`), followed by two `<space>`s or the end of a line; any  
38123 number of closing parenthesis (`' ) '`), closing brackets (`' ] '`), double quote (`' \" '`),  
38124 or single quote (`' ' '`) characters can appear between the punctuation mark and  
38125 the two `<space>`s or end-of-line

38126 In the remainder of the description of the *vi* utility, the term “buffer line” refers to a line in the  
38127 edit buffer and the term “display line” refers to the line or lines on the display screen used to  
38128 display one buffer line. The term “current line” refers to a specific “buffer line”.

38129 If there are display lines on the screen for which there are no corresponding buffer lines because  
38130 they correspond to lines that would be after the end of the file, they shall be displayed as a single  
38131 tilde (`' ~ '`) character, plus the terminating `<newline>`.

38132 The last line of the screen shall be used to report errors or display informational messages. It  
38133 shall also be used to display the input for “line-oriented commands” (`/`, `?`, `:`, and `!`). When a line-  
38134 oriented command is executed, the editor shall enter text input mode on the last line on the  
38135 screen, using the respective command characters as prompt characters. (In the case of the `!`  
38136 command, the associated motion shall be entered by the user before the editor enters text input  
38137 mode.) The line entered by the user shall be terminated by a `<newline>`, a non-`<control>`-V-  
38138 escaped `<carriage-return>`, or unescaped `<ESC>`. It is unspecified if more characters than  
38139 require a display width minus one column number of screen columns can be entered.

38140 If any command is executed that overwrites a portion of the screen other than the last line of the  
38141 screen (for example, the *ex* **suspend** or `!` commands), other than the *ex* **shell** command, the user  
38142 shall be prompted for a character before the screen is refreshed and the edit session continued.

38143 `<tab>`s shall take up the number of columns on the screen set by the **tabstop** edit option (see *ed*),  
38144 unless there are less than that number of columns before the display margin that will cause the  
38145 displayed line to be folded; in this case, they shall only take up the number of columns up to that  
38146 boundary.

38147 The cursor shall be placed on the current line and relative to the current column as specified by  
38148 each command described in the following sections.

38149 In open mode, if the current line is not already displayed, then it shall be displayed.

38150 In visual mode, if the current line is not displayed, then the lines that are displayed shall be  
38151 expanded, scrolled, or redrawn to cause an unspecified portion of the current line to be  
38152 displayed. If the screen is redrawn, no more than the number of display lines specified by the  
38153 value of the **window** edit option shall be displayed (unless the current line cannot be completely  
38154 displayed in the number of display lines specified by the **window** edit option) and the current  
38155 line shall be positioned as close to the center of the displayed lines as possible (within the  
38156 constraints imposed by the distance of the line from the beginning or end of the edit buffer). If  
38157 the current line is before the first line in the display and the screen is scrolled, an unspecified  
38158 portion of the current line shall be placed on the first line of the display. If the current line is after  
38159 the last line in the display and the screen is scrolled, an unspecified portion of the current line  
38160 shall be placed on the last line of the display.

38161 In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into  
38162 the lines at the bottom of the display that are available for its presentation, the editor may

38163 choose not to display any portion of the line. The lines of the display that do not contain text  
 38164 from the edit buffer for this reason shall each consist of a single '@' character.

38165 In visual mode, the editor may choose for unspecified reasons to not update lines in the display  
 38166 to correspond to the underlying edit buffer text. The lines of the display that do not correctly  
 38167 correspond to text from the edit buffer for this reason shall consist of a single '@' character  
 38168 (plus the terminating <newline>), and the <control>-R command shall cause the editor to  
 38169 update the screen to correctly represent the edit buffer.

38170 Open and visual mode commands that set the current column set it to a column position in the  
 38171 display, and not a character position in the line. In this case, however, the column position in the  
 38172 display shall be calculated for an infinite width display; for example, the column related to a  
 38173 character that is part of a line that has been folded onto additional screen lines will be offset from  
 38174 the display line column where the buffer line begins, not from the beginning of a particular  
 38175 display line.

38176 The display cursor column in the display is based on the value of the current column, as follows,  
 38177 with each rule applied in turn:

- 38178 1. If the current column is after the last display line column used by the displayed line, the  
 38179 display cursor column shall be set to the last display line column occupied by the last non-  
 38180 <newline> in the current line; otherwise, the display cursor column shall be set to the  
 38181 current column.
- 38182 2. If the character of which some portion is displayed in the display line column specified by  
 38183 the display cursor column requires more than a single display line column:
  - 38184 a. If in text input mode, the display cursor column shall be adjusted to the first display  
 38185 line column in which any portion of that character is displayed.
  - 38186 b. Otherwise, the display cursor column shall be adjusted to the last display line  
 38187 column in which any portion of that character is displayed.

38188 The current column shall not be changed by these adjustments to the display cursor column.

38189 If an error occurs during the parsing or execution of a vi command:

- 38190 • The terminal shall be alerted. Execution of the vi command shall stop, and the cursor (for  
 38191 example, the current line and column) shall not be further modified.
- 38192 • Unless otherwise specified by the following command sections, it is unspecified whether an  
 38193 informational message shall be displayed.
- 38194 • Any partially entered vi command shall be discarded.
- 38195 • If the vi command resulted from a **map** expansion, all characters from that **map** expansion  
 38196 shall be discarded, except as otherwise specified by the **map** command (see *ed*).
- 38197 • If the vi command resulted from the execution of a buffer, no further commands caused by  
 38198 the execution of the buffer shall be executed.

**Page Backwards**

*Synopsis:*     [*count*] <control>-B

If in open mode, the <control>-B command shall behave identically to the **z** command. Otherwise, if the current line is the first line of the edit buffer, it shall be an error.

If the **window** edit option is less than 3, display a screen where the last line of the display shall be some portion of:

(*current first line*) -1

otherwise, display a screen where the first line of the display shall be some portion of:

(*current first line*) - *count* x ((window edit option) -2)

If this calculation would result in a line that is before the first line of the edit buffer, the first line of the display shall display some portion of the first line of the edit buffer.

*Current line:* If no lines from the previous display remain on the screen, set to the last line of the display; otherwise, set to (*line* - the number of new lines displayed on this screen).

*Current column:* Set to non-<blank>.

**Scroll Forward**

*Synopsis:*     [*count*] <control>-D

If the current line is the last line of the edit buffer, it shall be an error.

If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D or <control>-U command. If there was no previous <control>-D or <control>-U command, *count* shall default to the value of the **scroll** edit option.

If in open mode, write lines starting with the line after the current line, until *count* lines or the last line of the file have been written.

*Current line:* If the current line + *count* is past the last line of the edit buffer, set to the last line of the edit buffer; otherwise, set to the current line + *count*.

*Current column:* Set to non-<blank>.

**Scroll Forward by Line**

*Synopsis:*     [*count*] <control>-E

Display the line *count* lines after the last line currently displayed.

If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines after the last line currently displayed, the last line of the display shall display some portion of the last line of the edit buffer.

*Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first line displayed.

*Current column:* Unchanged.

**Page Forward**

*Synopsis:*     [*count*] <control>-F

If in open mode, the <control>-F command shall behave identically to the **z** command. Otherwise, if the current line is the last line of the edit buffer, it shall be an error.

If the **window** edit option is less than 3, display a screen where the first line of the display shall be some portion of:

*(current last line) +1*

otherwise, display a screen where the first line of the display shall be some portion of:

*(current first line) + count x ((window edit option) -2)*

If this calculation would result in a line that is after the last line of the edit buffer, the last line of the display shall display some portion of the last line of the edit buffer.

*Current line:* If no lines from the previous display remain on the screen, set to the first line of the display; otherwise, set to *(line + the number of new lines displayed on this screen)*.

*Current column:* Set to non-<blank>.

**Display Information**

*Synopsis:*     <control>-G

This command shall be equivalent to the **ex file** command.

**Move Cursor Backwards**

*Synopsis:*     [*count*] <control>-H

              [*count*] h

              the current erase character (see stty)

If there are no characters before the current character on the current line, it shall be an error. If there are less than *count* previous characters on the current line, *count* shall be adjusted to the number of previous characters on the line.

If used as a motion command:

1. The text region shall be from the character before the starting cursor up to and including the *count*th character before the starting cursor.
2. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

*Current line:* Unchanged.

*Current column:* Set to *(column - the number of columns occupied by count characters ending with the previous current column)*.

**Move Down**

*Synopsis:*     [*count*] <newline>  
                  [*count*] <control>-J  
                  [*count*] <control>-M  
                  [*count*] <control>-N  
                  [*count*] j  
                  [*count*] <carriage-return>  
                  [*count*] +

If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

If used as a motion command:

1. The text region shall include the starting line and the next *count* – 1 lines.
2. Any text copied to a buffer shall be in line mode.

If not used as a motion command:

*Current line:* Set to *current line* + *count*.

*Current column:* Set to non-<blank> for the <carriage-return>, <control>-M, and + commands; otherwise, unchanged.

**Clear and Redisplay**

*Synopsis:*     <control>-L

If in open mode, clear the screen and redisplay the current line. Otherwise, clear and redisplay the screen.

*Current line:* Unchanged.

*Current column:* Unchanged.

**Move Up**

*Synopsis:*     [*count*] <control>-P  
                  [*count*] k  
                  [*count*] –

If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

If used as a motion command:

1. The text region shall include the starting line and the previous *count* lines.
2. Any text copied to a buffer shall be in line mode.

If not used as a motion command:

*Current line:* Set to *current line* – *count*.

*Current column:* Set to non-<blank> for the – command; otherwise, unchanged.

**Redraw Screen**

*Synopsis:*     <control>-R

If any lines have been deleted from the display screen and flagged as deleted on the terminal using the @ convention (see the beginning of the EXTENDED DESCRIPTION section), they shall be redisplayed to match the contents of the edit buffer.

It is unspecified whether lines flagged with @ because they do not fit on the terminal display shall be affected.

*Current line:* Unchanged.

*Current column:* Unchanged.

**Scroll Backward**

*Synopsis:*     [count] <control>-U

If the current line is the first line of the edit buffer, it shall be an error.

If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D or <control>-U command. If there was no previous <control>-D or <control>-U command, *count* shall default to the value of the **scroll** edit option.

*Current line:* If *count* is greater than the current line, set to 1; otherwise, set to the current line – *count*.

*Current column:* Set to non-<blank>.

**Scroll Backward by Line**

*Synopsis:*     [count] <control>-Y

Display the line *count* lines before the first line currently displayed.

If the current line is the first line of the edit buffer, it shall be an error. If this calculation would result in a line that is before the first line of the edit buffer, the first line of the display shall display some portion of the first line of the edit buffer.

*Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first line displayed.

*Current column:* Unchanged.

**Edit the Alternate File**

*Synopsis:*     <control>-^

This command shall be equivalent to the *ex* **edit** command, with the alternate pathname as its argument.

**Terminate Command or Input Mode**

*Synopsis:*     <ESC>

If a partial *vi* command (as defined by at least one, non-*count* character) has been entered, discard the *count* and the command character(s).

Otherwise, if no command characters have been entered, and the <ESC> was the result of a map expansion, the terminal shall be alerted and the <ESC> character shall be discarded, but it shall not be an error.



38336 Otherwise, it shall be an error.

38337 *Current line*: Unchanged.

38338 *Current column*: Unchanged.

### 38339 **Search for tagstring**

38340 *Synopsis*:     <control>-]

38341 If the current character is not a word or <blank>, it shall be an error.

38342 This command shall be equivalent to the *ex* **tag** command, with the argument to that command  
38343 defined as follows.

38344 If the current character is a <blank>:

- 38345     1. Skip all <blank>s after the cursor up to the end of the line.
- 38346     2. If the end of the line is reached, it shall be an error.

38347 Then, the argument to the *ex* **tag** command shall be the current character and all subsequent  
38348 characters, up to the first non-word character or the end of the line.

### 38349 **Move Cursor Forward**

38350 *Synopsis*:     [*count*] <space>  
38351               [*count*] 1 (ell)

38352 If there are less than *count* non-<newline>s after the cursor on the current line, *count* shall be  
38353 adjusted to the number of non-<newline>s after the cursor on the line.

38354 If used as a motion command:

- 38355     1. If the current or *count*th character after the cursor is the last non-<newline> in the line, the  
38356       text region shall be comprised of the current character up to and including the last non-  
38357       <newline> in the line. Otherwise, the text region shall be from the current character up to,  
38358       but not including, the *count*th character after the cursor.
- 38359     2. Any text copied to a buffer shall be in character mode.

38360 If not used as a motion command:

38361 If there are no non-<newline>s after the current character on the current line, it shall be an error.

38362 *Current line*: Unchanged.

38363 *Current column*: Set to the last column that displays any portion of the *count*th character after the  
38364 current character.

### 38365 **Replace Text with Results from Shell Command**

38366 *Synopsis*:     [*count*] ! *motion shell-commands* <newline>

38367 If the motion command is the ! command repeated:

- 38368     1. If the edit buffer is empty and no *count* was supplied, the command shall be the equivalent  
38369       of the *ex* **:read !** command, with the text input, and no text shall be copied to any buffer.
- 38370     2. Otherwise:
  - 38371       a. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be  
38372       an error.

38373                   b. The text region shall be from the current line up to and including the next *count* –1  
38374                   lines.

38375                   Otherwise, the text region shall be the lines in which any character of the text region specified by  
38376                   the motion command appear.

38377                   Any text copied to a buffer shall be in line mode.

38378                   This command shall be equivalent to the *ex !* command for the specified lines.

### 38379                   **Move Cursor to End-of-Line**

38380                   Synopsis:       [*count*] \$

38381                   It shall be an error if there are less than (*count* –1) lines after the current line in the edit buffer.

38382                   If used as a motion command:

- 38383                   1. If *count* is 1:
  - 38384                   a. It shall be an error if the line is empty.
  - 38385                   b. Otherwise, the text region shall consist of all characters from the starting cursor to  
38386                   the last non-<newline> in the line, inclusive, and any text copied to a buffer shall be  
38387                   in character mode.
- 38388                   2. Otherwise, if the starting cursor position is at or before the first non-<blank> in the line,  
38389                   the text region shall consist of the current and the next *count* –1 lines, and any text saved to  
38390                   a buffer shall be in line mode.
- 38391                   3. Otherwise, the text region shall consist of all characters from the starting cursor to the last  
38392                   non-<newline> in the line that is *count* –1 lines forward from the current line, and any text  
38393                   copied to a buffer shall be in character mode.

38394                   If not used as a motion command:

38395                   *Current line*: Set to the *current line* + *count* –1.

38396                   *Current column*: The current column is set to the last display line column of the last non-  
38397                   <newline> in the line, or column position 1 if the line is empty.

38398                   The current column shall be adjusted to be on the last display line column of the last non-  
38399                   <newline> of the current line as subsequent commands change the current line, until a  
38400                   command changes the current column.

### 38401                   **Move to Matching Character**

38402                   Synopsis:       %

38403                   If the character at the current position is not a parenthesis, bracket, or curly brace, search  
38404                   forward in the line to the first one of those characters. If no such character is found, it shall be an  
38405                   error.

38406                   The matching character shall be the parenthesis, bracket, or curly brace matching the  
38407                   parenthesis, bracket, or curly brace, respectively, that was at the current position or that was  
38408                   found on the current line.

38409                   Matching shall be determined as follows, for an open parenthesis:

- 38410                   1. Set a counter to 1.
- 38411                   2. Search forwards until a parenthesis is found or the end of the edit buffer is reached.

- 38412           3. If the end of the edit buffer is reached, it shall be an error.
- 38413           4. If an open parenthesis is found, increment the counter by 1.
- 38414           5. If a close parenthesis is found, decrement the counter by 1.
- 38415           6. If the counter is zero, the current character is the matching character.
- 38416       Matching for a close parenthesis shall be equivalent, except that the search shall be backwards,  
38417       from the starting character to the beginning of the buffer, a close parenthesis shall increment the  
38418       counter by 1, and an open parenthesis shall decrement the counter by 1.
- 38419       Matching for brackets and curly braces shall be equivalent, except that searching shall be done  
38420       for open and close brackets or open and close curly braces. It is implementation-defined whether  
38421       other characters are searched for and matched as well.
- 38422       If used as a motion command:
- 38423           1. If the matching cursor was after the starting cursor in the edit buffer, and the starting  
38424           cursor position was at or before the first non-`<blank>` non-`<newline>` in the starting line,  
38425           and the matching cursor position was at or after the last non-`<blank>` non-`<newline>` in  
38426           the matching line, the text region shall consist of the current line to the matching line,  
38427           inclusive, and any text copied to a buffer shall be in line mode.
- 38428           2. If the matching cursor was before the starting cursor in the edit buffer, and the starting  
38429           cursor position was at or after the last non-`<blank>` non-`<newline>` in the starting line, and  
38430           the matching cursor position was at or before the first non-`<blank>` non-`<newline>` in the  
38431           matching line, the text region shall consist of the current line to the matching line,  
38432           inclusive, and any text copied to a buffer shall be in line mode.
- 38433           3. Otherwise, the text region shall consist of the starting character to the matching character,  
38434           inclusive, and any text copied to a buffer shall be in character mode.
- 38435       If not used as a motion command:
- 38436       *Current line*: Set to the line where the matching character is located.
- 38437       *Current column*: Set to the last column where any portion of the matching character is displayed.
- 38438       **Repeat Substitution**
- 38439       Synopsis:       &
- 38440       Repeat the previous substitution command. This command shall be equivalent to the *ex* **&**  
38441       command with the current line as its addresses, and without *options*, *count*, or *flags*.
- 38442       **Return to Previous Context at Beginning of Line**
- 38443       Synopsis:       ' *character*
- 38444       It shall be an error if there is no line in the edit buffer marked by *character*.
- 38445       If used as a motion command:
- 38446           1. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
38447           and the marked cursor in the edit buffer shall be logically swapped.
- 38448           2. The text region shall consist of the starting line up to and including the marked line, and  
38449           any text copied to a buffer shall be in line mode.
- 38450       If not used as a motion command:

38451 *Current line*: Set to the line referenced by the mark.

38452 *Current column*: Set to non-<blank>.

### 38453 **Return to Previous Context**

38454 *Synopsis*:     \ *character*

38455 It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer  
38456 contains a character in the saved numbered character position, it shall be as if the marked  
38457 position is the first non-<blank>.

38458 If used as a motion command:

- 38459     1. It shall be an error if the marked cursor references the same character in the edit buffer as  
38460       the starting cursor.
- 38461     2. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
38462       and the marked cursor in the edit buffer shall be logically swapped.
- 38463     3. If the starting line is empty or the starting cursor is at or before the first non-<blank> non-  
38464       <newline> of the starting line, and the marked cursor line is empty or the marked cursor  
38465       references the first character of the marked cursor line, the text region shall consist of all  
38466       lines containing characters from the starting cursor to the line before the marked cursor  
38467       line, inclusive, and any text copied to a buffer shall be in line mode.
- 38468     4. Otherwise, if the marked cursor line is empty or the marked cursor references a character  
38469       at or before the first non-<blank> non-<newline> of the marked cursor line, the region of  
38470       text shall be from the starting cursor to the last non-<newline> of the line before the  
38471       marked cursor line, inclusive, and any text copied to a buffer shall be in character mode.
- 38472     5. Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked  
38473       cursor (exclusive), and any text copied to a buffer shall be in character mode.

38474 If not used as a motion command:

38475 *Current line*: Set to the line referenced by the mark.

38476 *Current column*: Set to the last column in which any portion of the character referenced by the  
38477 mark is displayed.

### 38478 **Return to Previous Section**

38479 *Synopsis*:     [*count*] [ [

38480 Move the cursor backward through the edit buffer to the first character of the previous section  
38481 boundary, *count* times.

38482 If used as a motion command:

- 38483     1. If the starting cursor was at the first character of the starting line or the starting line was  
38484       empty, and the first character of the boundary was the first character of the boundary line,  
38485       the text region shall consist of the current line up to and including the line where the  
38486       *count*th next boundary starts, and any text copied to a buffer shall be in line mode.
- 38487     2. If the boundary was the last line of the edit buffer or the last non-<newline> of the last line  
38488       of the edit buffer, the text region shall consist of the last character in the edit buffer up to  
38489       and including the starting character, and any text saved to a buffer shall be in character  
38490       mode.

38491 3. Otherwise, the text region shall consist of the starting character up to but not including the  
 38492 first character in the *countth* next boundary, and any text copied to a buffer shall be in  
 38493 character mode.

38494 If not used as a motion command:

38495 *Current line*: Set to the line where the *countth* next boundary in the edit buffer starts.

38496 *Current column*: Set to the last column in which any portion of the first character of the *countth*  
 38497 next boundary is displayed, or column position 1 if the line is empty.

### 38498 **Move to Next Section**

38499 *Synopsis*: [ *count* ] ]

38500 Move the cursor forward through the edit buffer to the first character of the next section  
 38501 boundary, *count* times.

38502 If used as a motion command:

- 38503 1. If the starting cursor was at the first character of the starting line or the starting line was  
 38504 empty, and the first character of the boundary was the first character of the boundary line,  
 38505 the text region shall consist of the current line up to and including the line where the  
 38506 *countth* previous boundary starts, and any text copied to a buffer shall be in line mode.
- 38507 2. If the boundary was the first line of the edit buffer, the text region shall consist of the first  
 38508 character in the edit buffer up to but not including the starting character, and any text  
 38509 copied to a buffer shall be in character mode.
- 38510 3. Otherwise, the text region shall consist of the first character in the *countth* previous section  
 38511 boundary up to but not including the starting character, and any text copied to a buffer  
 38512 shall be in character mode.

38513 If not used as a motion command:

38514 *Current line*: Set to the line where the *countth* previous boundary in the edit buffer starts.

38515 *Current column*: Set to the last column in which any portion of the first character of the *countth*  
 38516 previous boundary is displayed, or column position 1 if the line is empty.

### 38517 **Move to First Non-<blank> Position on Current Line**

38518 *Synopsis*: ^

38519 If used as a motion command:

- 38520 1. If the line has no non-<blank> non-<newline>s, or if the cursor is at the first non-<blank>  
 38521 non-<newline> of the line, it shall be an error.
- 38522 2. If the cursor is before the first non-<blank> non-<newline> of the line, the text region shall  
 38523 be comprised of the current character, up to, but not including, the first non-<blank> non-  
 38524 <newline> of the line.
- 38525 3. If the cursor is after the first non-<blank> non-<newline> of the line, the text region shall  
 38526 be from the character before the starting cursor up to and including the first non-<blank>  
 38527 non-<newline> of the line.
- 38528 4. Any text copied to a buffer shall be in character mode.

38529 If not used as a motion command:

38530 *Current line*: Unchanged.

38531 *Current column*: Set to non-<blank>.

### 38532 **Current and Line Above**

38533 *Synopsis*:     [*count*] \_

38534 If there are less than *count* − 1 lines after the current line in the edit buffer, it shall be an error.

38535 If used as a motion command:

- 38536     1. If *count* is less than 2, the text region shall be the current line.
- 38537     2. Otherwise, the text region shall include the starting line and the next *count* − 1 lines.
- 38538     3. Any text copied to a buffer shall be in line mode.

38539 If not used as a motion command:

38540 *Current line*: Set to current line + *count* − 1.

38541 *Current column*: Set to non-<blank>.

### 38542 **Move Back to Beginning of Sentence**

38543 *Synopsis*:     [*count*] (

38544 Move backward to the beginning of a sentence. This command shall be equivalent to the **[[**  
38545 command, with the exception that sentence boundaries shall be used instead of section  
38546 boundaries.

### 38547 **Move Forward to Beginning of Sentence**

38548 *Synopsis*:     [*count*] )

38549 Move forward to the beginning of a sentence. This command shall be equivalent to the **]]**  
38550 command, with the exception that sentence boundaries shall be used instead of section  
38551 boundaries.

### 38552 **Move Back to Preceding Paragraph**

38553 *Synopsis*:     [*count*] {

38554 Move back to the beginning of the preceding paragraph. This command shall be equivalent to  
38555 the **[[** command, with the exception that paragraph boundaries shall be used instead of section  
38556 boundaries.

### 38557 **Move Forward to Next Paragraph**

38558 *Synopsis*:     [*count*] }

38559 Move forward to the beginning of the next paragraph. This command shall be equivalent to the  
38560 **]]** command, with the exception that paragraph boundaries shall be used instead of section  
38561 boundaries.

**Move to Specific Column Position**

*Synopsis:*     [*count*] |

For the purposes of this command, lines that are too long for the current display and that have been folded shall be treated as having a single, 1-based, number of columns.

If there are less than *count* columns in which characters from the current line are displayed on the screen, *count* shall be adjusted to be the last column in which any portion of the line is displayed on the screen.

If used as a motion command:

1. If the line is empty, or the cursor character is the same as the character on the *count*th column of the line, it shall be an error.
2. If the cursor is before the *count*th column of the line, the text region shall be comprised of the current character, up to but not including the character on the *count*th column of the line.
3. If the cursor is after the *count*th column of the line, the text region shall be from the character before the starting cursor up to and including the character on the *count*th column of the line.
4. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

*Current line:* Unchanged.

*Current column:* Set to the last column in which any portion of the character that is displayed in the *count* column of the line is displayed.

**Reverse Find Character**

*Synopsis:*     [*count*] ,

If the last **F**, **f**, **T**, or **t** command was **F**, **f**, **T**, or **t**, this command shall be equivalent to an **f**, **F**, **t**, or **T** command, respectively, with the specified *count* and the same search character.

If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.

**Repeat**

*Synopsis:*     [*count*] .

Repeat the last **!**, **<**, **>**, **A**, **C**, **D**, **I**, **J**, **O**, **P**, **R**, **S**, **X**, **Y**, **a**, **c**, **d**, **i**, **o**, **p**, **r**, **s**, **x**, **y**, or **~** command. It shall be an error if none of these commands have been executed. Commands (other than commands that enter text input mode) executed as a result of map expansions, shall not change the value of the last repeatable command.

Repeated commands with associated motion commands shall repeat the motion command as well; however, any specified *count* shall replace the *count*(s) that were originally specified to the repeated command or its associated motion command.

If the motion component of the repeated command is **f**, **F**, **t**, or **T**, the repeated command shall not set the remembered search character for the **;** and **,** commands.

If the repeated command is **p** or **P**, and the buffer associated with that command was a numeric buffer named with a number less than 9, the buffer associated with the repeated command shall be set to be the buffer named by the name of the previous buffer logically incremented by 1.

38602 If the repeated character is a text input command, the input text associated with that command  
 38603 is repeated literally:

- 38604 • Input characters are neither macro or abbreviation-expanded.
- 38605 • Input characters are not interpreted in any special way with the exception that <newline>,  
 38606 <carriage-return>, and <control>-T behave as described in **Input Mode Commands in vi** (on  
 38607 page 1022).

38608 *Current line*: Set as described for the repeated command.

38609 *Current column*: Set as described for the repeated command.

## 38610 **Find Regular Expression**

38611 *Synopsis*:        /

38612 If the input line contains no non-<newline>s, it shall be equivalent to a line containing only the  
 38613 last regular expression encountered. The enhanced regular expressions supported by vi are  
 38614 described in **Regular Expressions in ex** (on page 391).

38615 Otherwise, the line shall be interpreted as one or more regular expressions, optionally followed  
 38616 by an address offset or a vi **z** command.

38617 If the regular expression is not the last regular expression on the line, or if a line offset or **z**  
 38618 command is specified, the regular expression shall be terminated by an unescaped '/'  
 38619 character, which shall not be used as part of the regular expression. If the regular expression is  
 38620 not the first regular expression on the line, it shall be preceded by zero or more <blank>s, a  
 38621 semicolon, zero or more <blank>s, and a leading '/' character, which shall not be interpreted as  
 38622 part of the regular expression. It shall be an error to precede any regular expression with any  
 38623 characters other than these.

38624 Each search shall begin from the character after the first character of the last match (or, if it is the  
 38625 first search, after the cursor). If the **wrapsan** edit option is set, the search shall continue to the  
 38626 character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be  
 38627 an error if any search fails to find a match, and an informational message to this effect shall be  
 38628 displayed.

38629 An optional address offset (see **Addressing in ex** (on page 361)) can be specified after the last  
 38630 regular expression by including a trailing '/' character after the regular expression and  
 38631 specifying the address offset. This offset will be from the line containing the match for the last  
 38632 regular expression specified. It shall be an error if the line offset would indicate a line address  
 38633 less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be  
 38634 supported. It shall be an error to follow the address offset with any other characters than  
 38635 <blank>s.

38636 If not used as a motion command, an optional **z** command (see **Redraw Window** (on page 1021))  
 38637 can be specified after the last regular expression by including a trailing '/' character after the  
 38638 regular expression, zero or more <blank>s, a 'z', zero or more <blank>s, an optional new  
 38639 **window** edit option value, zero or more <blank>s, and a location character. The effect shall be as  
 38640 if the **z** command was executed after the / command. It shall be an error to follow the **z**  
 38641 command with any other characters than <blank>s.

38642 The remembered search direction shall be set to forward.

38643 If used as a motion command:

- 38644 1. It shall be an error if the last match references the same character in the edit buffer as the  
 38645 starting cursor.



- 38646 2. If any address offset is specified, the last match shall be adjusted by the specified offset as  
38647 described previously.
- 38648 3. If the starting cursor is after the last match, then the locations of the starting cursor and the  
38649 last match in the edit buffer shall be logically swapped.
- 38650 4. If any address offset is specified, the text region shall consist of all lines containing  
38651 characters from the starting cursor to the last match line, inclusive, and any text copied to a  
38652 buffer shall be in line mode.
- 38653 5. Otherwise, if the starting line is empty or the starting cursor is at or before the first non-  
38654 <blank> non-<newline> of the starting line, and the last match line is empty or the last  
38655 match starts at the first character of the last match line, the text region shall consist of all  
38656 lines containing characters from the starting cursor to the line before the last match line,  
38657 inclusive, and any text copied to a buffer shall be in line mode.
- 38658 6. Otherwise, if the last match line is empty or the last match begins at a character at or  
38659 before the first non-<blank> non-<newline> of the last match line, the region of text shall  
38660 be from the current cursor to the last non-<newline> of the line before the last match line,  
38661 inclusive, and any text copied to a buffer shall be in character mode.
- 38662 7. Otherwise, the region of text shall be from the current cursor (inclusive), to the first  
38663 character of the last match (exclusive), and any text copied to a buffer shall be in character  
38664 mode.

38665 If not used as a motion command:

38666 *Current line:* If a match is found, set to the last matched line plus the address offset, if any;  
38667 otherwise, unchanged.

38668 *Current column:* Set to the last column on which any portion of the first character in the last  
38669 matched string is displayed, if a match is found; otherwise, unchanged.

## 38670 **Move to First Character in Line**

38671 *Synopsis:*     0    (zero)

38672 Move to the first character on the current line. The character '0' shall not be interpreted as a  
38673 command if it is immediately preceded by a digit.

38674 If used as a motion command:

- 38675 1. If the cursor character is the first character in the line, it shall be an error.
- 38676 2. The text region shall be from the character before the cursor character up to and including  
38677 the first character in the line.
- 38678 3. Any text copied to a buffer shall be in character mode.

38679 If not used as a motion command:

38680 *Current line:* Unchanged.

38681 *Current column:* The last column in which any portion of the first character in the line is  
38682 displayed, or if the line is empty, unchanged.

**Execute an ex Command**

*Synopsis:*       :

Execute one or more *ex* commands.

If any portion of the screen other than the last line of the screen was overwritten by any *ex* command (except **shell**), *vi* shall display a message indicating that it is waiting for an input from the user, and shall then read a character. This action may also be taken for other, unspecified reasons.

If the next character entered is a ' : ', another *ex* command shall be accepted and executed. Any other character shall cause the screen to be refreshed and *vi* shall return to command mode.

*Current line:* As specified for the *ex* command.

*Current column:* As specified for the *ex* command.

**Repeat Find**

*Synopsis:*       [*count*] ;

This command shall be equivalent to the last **F**, **f**, **T**, or **t** command, with the specified *count*, and with the same search character used for the last **F**, **f**, **T**, or **t** command. If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.

**Shift Left**

*Synopsis:*       [*count*] < *motion*

If the motion command is the < command repeated:

1. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an error.
2. The text region shall be from the current line, up to and including the next *count* -1 lines.

Shift any line in the text region specified by the *count* and motion command one shiftwidth (see the *ex* **shiftwidth** option) toward the start of the line, as described by the *ex* < command. The unshifted lines shall be copied to the unnamed buffer in line mode.

*Current line:* If the motion was from the current cursor position toward the end of the edit buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region specified by the motion command.

*Current column:* Set to non-<blank>.

**Shift Right**

*Synopsis:*       [*count*] > *motion*

If the motion command is the > command repeated:

1. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an error.
2. The text region shall be from the current line, up to and including the next *count* -1 lines.

Shift any line with characters in the text region specified by the *count* and motion command one shiftwidth (see the *ex* **shiftwidth** option) away from the start of the line, as described by the *ex* > command. The unshifted lines shall be copied into the unnamed buffer in line mode.

38721 *Current line:* If the motion was from the current cursor position toward the end of the edit  
 38722 buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
 38723 specified by the motion command.

38724 *Current column:* Set to non-<blank>.

## 38725 **Scan Backwards for Regular Expression**

38726 *Synopsis:*     ?

38727 Scan backwards; the ? command shall be equivalent to the / command (see **Find Regular**  
 38728 **Expression** (on page 1004)) with the following exceptions:

- 38729     1. The input prompt shall be a ' ? '.
- 38730     2. Each search shall begin from the character before the first character of the last match (or, if  
 38731 it is the first search, the character before the cursor character).
- 38732     3. The search direction shall be from the cursor toward the beginning of the edit buffer, and  
 38733 the **wrapsan** edit option shall affect whether the search wraps to the end of the edit buffer  
 38734 and continues.
- 38735     4. The remembered search direction shall be set to backward.

## 38736 **Execute**

38737 *Synopsis:*     @*buffer*

38738 If the *buffer* is specified as @, the last buffer executed shall be used. If no previous buffer has been  
 38739 executed, it shall be an error.

38740 Behave as if the contents of the named buffer were entered as standard input. After each line of a  
 38741 line-mode buffer, and all but the last line of a character mode buffer, behave as if a <newline>  
 38742 were entered as standard input.

38743 If an error occurs during this process, an error message shall be written, and no more characters  
 38744 resulting from the execution of this command shall be processed.

38745 If a *count* is specified, behave as if that count were entered as user input before the characters  
 38746 from the @ buffer were entered.

38747 *Current line:* As specified for the individual commands.

38748 *Current column:* As specified for the individual commands.

## 38749 **Reverse Case**

38750 *Synopsis:*     [*count*] ~

38751 Reverse the case of the current character and the next *count* –1 characters, such that lowercase  
 38752 characters that have uppercase counterparts shall be changed to uppercase characters, and  
 38753 uppercase characters that have lowercase counterparts shall be changed to lowercase characters,  
 38754 as prescribed by the current locale. No other characters shall be affected by this command.

38755 If there are less than *count* –1 characters after the cursor in the edit buffer, *count* shall be adjusted  
 38756 to the number of characters after the cursor in the edit buffer minus 1.

38757 For the purposes of this command, the next character after the last non-<newline> on the line  
 38758 shall be the next character in the edit buffer.

38759 *Current line:* Set to the line including the (*count*–1)th character after the cursor.

38760 *Current column*: Set to the last column in which any portion of the (*count*–1)th character after the  
 38761 cursor is displayed.

## 38762 **Append**

38763 *Synopsis*:     [*count*] a

38764 Enter text input mode after the current cursor position. No characters already in the edit buffer  
 38765 shall be affected by this command. A *count* shall cause the input text to be appended *count* –1  
 38766 more times to the end of the input.

38767 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**  
 38768 (on page 1022)).

## 38769 **Append at End-of-Line**

38770 *Synopsis*:     [*count*] A

38771 This command shall be equivalent to the *vi* command:

38772 \$ [ *count* ] a

38773 (see **Append**).

## 38774 **Move Backward to Preceding Word**

38775 *Synopsis*:     [*count*] b

38776 With the exception that words are used as the delimiter instead of bigwords, this command shall  
 38777 be equivalent to the **B** command.

## 38778 **Move Backward to Preceding Bigword**

38779 *Synopsis*:     [*count*] B

38780 If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an  
 38781 error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count*  
 38782 shall be adjusted to the number of bigword beginnings between the cursor and the start of the  
 38783 edit buffer.

38784 If used as a motion command:

- 38785     1. The text region shall be from the first character of the *count*th previous bigword beginning  
 38786       up to but not including the cursor character.
- 38787     2. Any text copied to a buffer shall be in character mode.

38788 If not used as a motion command:

38789 *Current line*: Set to the line containing the *current column*.

38790 *Current column*: Set to the last column upon which any part of the first character of the *count*th  
 38791 previous bigword is displayed.

**Change**

*Synopsis:*     [*buffer*] [*count*] *c motion*

If the motion command is the **c** command repeated:

1. The buffer text shall be in line mode.
2. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an error.
3. The text region shall be from the current line up to and including the next *count* - 1 lines.

Otherwise, the buffer text mode and text region shall be as specified by the motion command.

The replaced text shall be copied into *buffer*, if specified, and into the unnamed buffer. If the text to be replaced contains characters from more than a single line, or the buffer text is in line mode, the replaced text shall be copied into the numeric buffers as well.

If the buffer text is in line mode:

1. Any lines that contain characters in the region shall be deleted, and the editor shall enter text input mode at the beginning of a new line which shall replace the first line deleted.
2. If the **autoindent** edit option is set, **autoindent** characters equal to the **autoindent** characters on the first line deleted shall be inserted as if entered by the user.

Otherwise, if characters from more than one line are in the region of text:

1. The text shall be deleted.
2. Any text remaining in the last line in the text region shall be appended to the first line in the region, and the last line in the region shall be deleted.
3. The editor shall enter text input mode after the last character not deleted from the first line in the text region, if any; otherwise, on the first column of the first line in the region.

Otherwise:

1. If the glyph for ' \$ ' is smaller than the region, the end of the region shall be marked with a ' \$ '.
2. The editor shall enter text input mode, overwriting the region of text.

*Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi** (on page 1022)).

**Change to End-of-Line**

*Synopsis:*     [*buffer*] [*count*] *C*

This command shall be equivalent to the *vi* command:

[*buffer*] [*count*] *c\$*

See the **c** command.

**Delete**

*Synopsis:*     [*buffer*] [*count*] d *motion*

If the motion command is the **d** command repeated:

1. The buffer text shall be in line mode.
2. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an error.
3. The text region shall be from the current line up to and including the next *count* -1 lines.

Otherwise, the buffer text mode and text region shall be as specified by the motion command.

If in open mode, and the current line is deleted, and the line remains on the display, an '@' character shall be displayed as the first glyph of that line.

Delete the region of text into *buffer*, if specified, and into the unnamed buffer. If the text to be deleted contains characters from more than a single line, or the buffer text is in line mode, the deleted text shall be copied into the numeric buffers, as well.

*Current line:* Set to the first text region line that appears in the edit buffer, unless that line has been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit buffer is empty.

*Current column:*

1. If the line is empty, set to column position 1.
2. Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the end of the edit buffer:
  - a. If a character from the current line is displayed in the current column, set to the last column that displays any portion of that character.
  - b. Otherwise, set to the last column in which any portion of any character in the line is displayed.
3. Otherwise, if a character is displayed in the column that began the text region, set to the last column that displays any portion of that character.
4. Otherwise, set to the last column in which any portion of any character in the line is displayed.

**Delete to End-of-Line**

*Synopsis:*     [*buffer*] D

Delete the text from the current position to the end of the current line; equivalent to the *vi* command:

[*buffer*] d\$

### 38858 **Move to End-of-Word**

38859 *Synopsis:*     [*count*] e

38860 With the exception that words are used instead of bigwords as the delimiter, this command shall  
38861 be equivalent to the **E** command.

### 38862 **Move to End-of-Bigword**

38863 *Synopsis:*     [*count*] E

38864 If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor  
38865 and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between  
38866 the cursor and the end of the edit buffer.

38867 If used as a motion command:

- 38868     1. The text region shall be from the last character of the *count*th next bigword up to and  
38869       including the cursor character.
- 38870     2. Any text copied to a buffer shall be in character mode.

38871 If not used as a motion command:

38872 *Current line:* Set to the line containing the current column.

38873 *Current column:* Set to the last column upon which any part of the last character of the *count*th  
38874 next bigword is displayed.

### 38875 **Find Character in Current Line (Forward)**

38876 *Synopsis:*     [*count*] f *character*

38877 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

38878 If used as a motion command:

- 38879     1. The text range shall be from the cursor character up to and including the *count*th  
38880       occurrence of the specified character after the cursor.
- 38881     2. Any text copied to a buffer shall be in character mode.

38882 If not used as a motion command:

38883 *Current line:* Unchanged.

38884 *Current column:* Set to the last column in which any portion of the *count*th occurrence of the  
38885 specified character after the cursor appears in the line.

### 38886 **Find Character in Current Line (Reverse)**

38887 *Synopsis:*     [*count*] F *character*

38888 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

38889 If used as a motion command:

- 38890     1. The text region shall be from the *count*th occurrence of the specified character before the  
38891       cursor, up to, but not including the cursor character.
- 38892     2. Any text copied to a buffer shall be in character mode.

38893 If not used as a motion command:

38894 *Current line*: Unchanged.

38895 *Current column*: Set to the last column in which any portion of the *count*th occurrence of the  
38896 specified character before the cursor appears in the line.

### 38897 **Move to Line**

38898 *Synopsis*:     [*count*] G

38899 If *count* is not specified, it shall default to the last line of the edit buffer. If *count* is greater than  
38900 the last line of the edit buffer, it shall be an error.

38901 If used as a motion command:

- 38902     1. The text region shall be from the cursor line up to and including the specified line.
- 38903     2. Any text copied to a buffer shall be in line mode.

38904 If not used as a motion command:

38905 *Current line*: Set to *count* if *count* is specified; otherwise, the last line.

38906 *Current column*: Set to non-<blank>.

### 38907 **Move to Top of Screen**

38908 *Synopsis*:     [*count*] H

38909 If the beginning of the line *count* greater than the first line of which any portion appears on the  
38910 display does not exist, it shall be an error.

38911 If used as a motion command:

- 38912     1. If in open mode, the text region shall be the current line.
- 38913     2. Otherwise, the text region shall be from the starting line up to and including (the first line  
38914       of the display + *count* - 1).
- 38915     3. Any text copied to a buffer shall be in line mode.

38916 If not used as a motion command:

38917 If in open mode, this command shall set the current column to non-<blank> and do nothing else.

38918 Otherwise, it shall set the current line and current column as follows.

38919 *Current line*: Set to (the first line of the display + *count* - 1).

38920 *Current column*: Set to non-<blank>.

### 38921 **Insert Before Cursor**

38922 *Synopsis*:     [*count*] i

38923 Enter text input mode before the current cursor position. No characters already in the edit buffer  
38924 shall be affected by this command. A *count* shall cause the input text to be appended *count* - 1  
38925 more times to the end of the input.

38926 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**  
38927 (on page 1022)).



**Insert at Beginning of Line**

*Synopsis:*     [*count*] I

This command shall be equivalent to the *vi* command `^[count]i`.

**Join**

*Synopsis:*     [*count*] J

If the current line is the last line in the edit buffer, it shall be an error.

This command shall be equivalent to the *ex* **join** command with no addresses, and an *ex* command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an *ex* command *count* value of *count* -1 for any other value of *count*, except that the current line and column shall be set as follows.

*Current line:* Unchanged.

*Current column:* The last column in which any portion of the character following the last character in the initial line is displayed, or the last non-<newline> in the line if no characters were appended.

**Move to Bottom of Screen**

*Synopsis:*     [*count*] L

If the beginning of the line *count* less than the last line of which any portion appears on the display does not exist, it shall be an error.

If used as a motion command:

1. If in open mode, the text region shall be the current line.
2. Otherwise, the text region shall include all lines from the starting cursor line to (the last line of the display -(*count* -1)).
3. Any text copied to a buffer shall be in line mode.

If not used as a motion command:

1. If in open mode, this command shall set the current column to non-<blank> and do nothing else.
2. Otherwise, it shall set the current line and current column as follows.

*Current line:* Set to (the last line of the display -(*count* -1)).

*Current column:* Set to non-<blank>.

**Mark Position**

*Synopsis:*     m *letter*

This command shall be equivalent to the *ex* **mark** command with the specified character as an argument.

**Move to Middle of Screen**

*Synopsis:*     M

The middle line of the display shall be calculated as follows:

$(\text{the top line of the display}) + ((\text{number of lines displayed}) + 1) / 2) - 1$

If used as a motion command:

1. If in open mode, the text region shall be the current line.
2. Otherwise, the text region shall include all lines from the starting cursor line up to and including the middle line of the display.
3. Any text copied to a buffer shall be in line mode.

If not used as a motion command:

If in open mode, this command shall set the current column to non-<blank> and do nothing else.

Otherwise, it shall set the current line and current column as follows.

*Current line:* Set to the middle line of the display.

*Current column:* Set to non-<blank>.

**Repeat Regular Expression Find (Forward)**

*Synopsis:*     n

If the remembered search direction was forward, the **n** command shall be equivalent to the *vi* **/** command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* **?** command with no characters entered by the user.

If the **n** command is used as a motion command for the **!** command, the editor shall not enter text input mode on the last line on the screen, and shall behave as if the user entered a single '!' character as the text input.

**Repeat Regular Expression Find (Reverse)**

*Synopsis:*     N

Scan for the next match of the last pattern given to **/** or **?**, but in the reverse direction; this is the reverse of **n**.

If the remembered search direction was forward, the **N** command shall be equivalent to the *vi* **?** command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* **/** command with no characters entered by the user. If the **N** command is used as a motion command for the **!** command, the editor shall not enter text input mode on the last line on the screen, and shall behave as if the user entered a single **!** character as the text input.

**Insert Empty Line Below**

*Synopsis:*     o

Enter text input mode in a new line appended after the current line. A *count* shall cause the input text to be appended *count* - 1 more times to the end of the already added text, each time starting on a new, appended line.

*Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi** (on page 1022)).

**Insert Empty Line Above**

*Synopsis:*     ○

Enter text input mode in a new line inserted before the current line. A *count* shall cause the input text to be appended *count* –1 more times to the end of the already added text, each time starting on a new, appended line.

*Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi** (on page 1022)).

**Put from Buffer Following**

*Synopsis:*     [*buffer*] p

If no *buffer* is specified, the unnamed buffer shall be used.

If the buffer text is in line mode, the text shall be appended below the current line, and each line of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the already added text, each time starting on a new, appended line.

If the buffer text is in character mode, the text shall be appended into the current line after the cursor, and each line of the buffer other than the first and last shall become a new line in the edit buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the already added text, each time starting after the last added character.

*Current line:* If the buffer text is in line mode, set the line to line +1; otherwise, unchanged.

*Current column:* If the buffer text is in line mode:

1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any portion of the first non-<blank> in the line is displayed.
2. If there is no non-<blank> in the first line of the buffer, set to the last column on which any portion of the last non-<newline> in the first line of the buffer is displayed.

If the buffer text is in character mode:

1. If the text in the buffer is from more than a single line, then set to the last column on which any portion of the first character from the buffer is displayed.
2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any portion of the last character from the buffer is displayed.
3. Otherwise, set to the first column on which any portion of the first character from the buffer is displayed.

**Put from Buffer Before**

*Synopsis:*     [*buffer*] P

If no *buffer* is specified, the unnamed buffer shall be used.

If the buffer text is in line mode, the text shall be inserted above the current line, and each line of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the already added text, each time starting on a new, appended line.

If the buffer text is in character mode, the text shall be inserted into the current line before the cursor, and each line of the buffer other than the first and last shall become a new line in the edit buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the

39040 already added text, each time starting after the last added character.

39041 *Current line:* Unchanged.

39042 *Current column:* If the buffer text is in line mode:

39043 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any

39044 portion of that character is displayed.

39045 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which any

39046 portion of the last non-<newline> in the first line of the buffer is displayed.

39047 If the buffer text is in character mode:

39048 1. If the buffer is the unnamed buffer, set to the last column on which any portion of the last

39049 character from the buffer is displayed.

39050 2. Otherwise, set to the first column on which any portion of the first character from the

39051 buffer is displayed.

39052 **Enter ex Mode**

39053 *Synopsis:*     Q

39054 Leave visual or open mode and enter *ex* command mode.

39055 *Current line:* Unchanged.

39056 *Current column:* Unchanged.

39057 **Replace Character**

39058 *Synopsis:*     [count] r character

39059 Replace the *count* characters at and after the cursor with the specified character. If there are less

39060 than *count* non-<newline>s at and after the cursor on the line, it shall be an error.

39061 If character is <control>-V, any next character other than the <newline> shall be stripped of any

39062 special meaning and used as a literal character.

39063 If character is <ESC>, no replacement shall be made and the current line and current column

39064 shall be unchanged.

39065 If character is <carriage-return> or <newline>, *count* new lines shall be appended to the current

39066 line. All but the last of these lines shall be empty. *count* characters at and after the cursor shall be

39067 discarded, and any remaining characters after the cursor in the current line shall be moved to the

39068 last of the new lines. If the **autoindent** edit option is set, they shall be preceded by the same

39069 number of **autoindent** characters found on the line from which the command was executed.

39070 *Current line:* Unchanged unless the replacement character is a <carriage-return> or <newline>,

39071 in which case it shall be set to line + *count*.

39072 *Current column:* Set to the last column position on which a portion of the last replaced character

39073 is displayed, or if the replacement character caused new lines to be created, set to non-<blank>.

39074 **Replace Characters**39075 *Synopsis:*     R

39076 Enter text input mode at the current cursor position possibly replacing text on the current line. A  
 39077 *count* shall cause the input text to be appended *count* – 1 more times to the end of the input.

39078 *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
 39079 (on page 1022)).

39080 **Substitute Character**39081 *Synopsis:*     [*buffer*] [*count*] s

39082 This command shall be equivalent to the vi command:

39083 [*buffer*] [*count*] c<space>39084 **Substitute Lines**39085 *Synopsis:*     [*buffer*] [*count*] S

39086 This command shall be equivalent to the vi command:

39087 [*buffer*] [*count*] c\_39088 **Move Cursor to Before Character (Forward)**39089 *Synopsis:*     [*count*] t *character*39090 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

39091 If used as a motion command:

39092     1. The text region shall be from the cursor up to but not including the *count*th occurrence of  
 39093       the specified character after the cursor.

39094     2. Any text copied to a buffer shall be in character mode.

39095 If not used as a motion command:

39096 *Current line:* Unchanged.

39097 *Current column:* Set to the last column in which any portion of the character before the *count*th  
 39098 occurrence of the specified character after the cursor appears in the line.

39099 **Move Cursor to After Character (Reverse)**39100 *Synopsis:*     [*count*] T *character*39101 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

39102 If used as a motion command:

39103     1. If the character before the cursor is the specified character, it shall be an error.

39104     2. The text region shall be from the character before the cursor up to but not including the  
 39105       *count*th occurrence of the specified character before the cursor.

39106     3. Any text copied to a buffer shall be in character mode.

39107 If not used as a motion command:

39108 *Current line:* Unchanged.

39109 *Current column:* Set to the last column in which any portion of the character after the *count*th  
39110 occurrence of the specified character before the cursor appears in the line.

## 39111 **Undo**

39112 *Synopsis:*     u

39113 This command shall be equivalent to the *ex* **undo** command except that the current line and  
39114 current column shall be set as follows:

39115 *Current line:* Set to the first line added or changed if any; otherwise, move to the line preceding  
39116 any deleted text if one exists; otherwise, move to line 1.

39117 *Current column:* If undoing an *ex* command, set to the first non-<blank>.

39118 Otherwise, if undoing a text input command:

- 39119     1. If the command was a **C**, **c**, **O**, **o**, **R**, **S**, or **s** command, the current column shall be set to the  
39120 value it held when the text input command was entered.
- 39121     2. Otherwise, set to the last column in which any portion of the first character after the  
39122 deleted text is displayed, or, if no non-<newline>s follow the text deleted from this line, set  
39123 to the last column in which any portion of the last non-<newline> in the line is displayed,  
39124 or 1 if the line is empty.

39125 Otherwise, if a single line was modified (that is, not added or deleted) by the **u** command:

- 39126     1. If text was added or changed, set to the last column in which any portion of the first  
39127 character added or changed is displayed.
- 39128     2. If text was deleted, set to the last column in which any portion of the first character after  
39129 the deleted text is displayed, or, if no non-<newline>s follow the deleted text, set to the last  
39130 column in which any portion of the last non-<newline> in the line is displayed, or 1 if the  
39131 line is empty.

39132 Otherwise, set to non-<blank>.

## 39133 **Undo Current Line**

39134 *Synopsis:*     U

39135 Restore the current line to its state immediately before the most recent time that it became the  
39136 current line.

39137 *Current line:* Unchanged.

39138 *Current column:* Set to the first column in the line in which any portion of the first character in  
39139 the line is displayed.

## 39140 **Move to Beginning of Word**

39141 *Synopsis:*     [count] w

39142 With the exception that words are used as the delimiter instead of bigwords, this command shall  
39143 be equivalent to the **W** command.

**Move to Beginning of Bigword**

*Synopsis:*     [*count*] W

If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last bigword in the edit buffer.

If used as a motion command:

1. If the associated command is **c**, *count* is 1, and the cursor is on a <blank>, the region of text shall be the current character and no further action shall be taken.
2. If there are less than *count* bigwords between the cursor and the end of the edit buffer, then the command shall succeed, and the region of text shall include the last character of the edit buffer.
3. If there are <blank>s or an end-of-line that precede the *count*th bigword, and the associated command is **c**, the region of text shall be up to and including the last character before the preceding <blank>s or end-of-line.
4. If there are <blank>s or an end-of-line that precede the bigword, and the associated command is **d** or **y**, the region of text shall be up to and including the last <blank> before the start of the bigword or end-of-line.
5. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

1. If the cursor is on the last character of the edit buffer, it shall be an error.

*Current line:* Set to the line containing the current column.

*Current column:* Set to the last column in which any part of the first character of the *count*th next bigword is displayed.

**Delete Character at Cursor**

*Synopsis:*     [*buffer*] [*count*] x

Delete the *count* characters at and after the current character into *buffer*, if specified, and into the unnamed buffer.

If the line is empty, it shall be an error. If there are less than *count* non-<newline>s at and after the cursor on the current line, *count* shall be adjusted to the number of non-<newline>s at and after the cursor.

*Current line:* Unchanged.

*Current column:* If the line is empty, set to column position 1. Otherwise, if there were *count* or less non-<newline>s at and after the cursor on the current line, set to the last column that displays any part of the last non-<newline> of the line. Otherwise, unchanged.

**Delete Character Before Cursor**

*Synopsis:*     [*buffer*] [*count*] X

Delete the *count* characters before the current character into *buffer*, if specified, and into the unnamed buffer.

If there are no characters before the current character on the current line, it shall be an error. If there are less than *count* previous characters on the current line, *count* shall be adjusted to the number of previous characters on the line.

*Current line:* Unchanged.

*Current column:* Set to (current column – the width of the deleted characters).

**Yank**

*Synopsis:*     [*buffer*] [*count*] y *motion*

Copy (yank) the region of text into *buffer*, if specified, and into the unnamed buffer.

If the motion command is the **y** command repeated:

1. The buffer shall be in line mode.
2. If there are less than *count* – 1 lines after the current line in the edit buffer, it shall be an error.
3. The text region shall be from the current line up to and including the next *count* – 1 lines.

Otherwise, the buffer text mode and text region shall be as specified by the motion command.

*Current line:* If the motion was from the current cursor position toward the end of the edit buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region specified by the motion command.

*Current column:*

1. If the motion was from the current cursor position toward the end of the edit buffer, unchanged.
2. Otherwise, if the current line is empty, set to column position 1.
3. Otherwise, set to the last column that displays any part of the first character in the file that is part of the text region specified by the motion command.

**Yank Current Line**

*Synopsis:*     [*buffer*] [*count*] Y

This command shall be equivalent to the **vi** command:

[*buffer*] [*count*] y\_



39209 **Redraw Window**39210 If in open mode, the **z** command shall have the Synopsis:39211 *Synopsis:*     [*count*] **z**

39212 If *count* is not specified, it shall default to the **window** edit option -1. The **z** command shall be  
 39213 equivalent to the *ex z* command, with a type character of = and a *count* of *count* -2, except that  
 39214 the current line and current column shall be set as follows, and the **window** edit option shall not  
 39215 be affected. If the calculation for the *count* argument would result in a negative number, the  
 39216 *count* argument to the *ex z* command shall be zero. A blank line shall be written after the last line  
 39217 is written.

39218 *Current line:* Unchanged.39219 *Current column:* Unchanged.39220 If not in open mode, the **z** command shall have the following Synopsis:39221 *Synopsis:*     [*line*] **z** [*count*] *character*

39222 If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the  
 39223 number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

39224 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in the  
 39225 *ex window* command), and the screen shall be redrawn.

39226 *line* shall be placed as specified by the following characters:

39227 &lt;newline&gt;, &lt;carriage-return&gt;

39228     Place the beginning of the line on the first line of the display.

39229 .     Place the beginning of the line in the center of the display. The middle line of the display  
 39230 shall be calculated as described for the **M** command.

39231 -     Place an unspecified portion of the line on the last line of the display.

39232 +     If *line* was specified, equivalent to the <newline> case. If *line* was not specified, display a  
 39233 screen where the first line of the display shall be (current last line) +1. If there are no lines  
 39234 after the last line in the display, it shall be an error.

39235 ^     If *line* was specified, display a screen where the last line of the display shall contain an  
 39236 unspecified portion of the first line of a display that had an unspecified portion of the  
 39237 specified line on the last line of the display. If this calculation results in a line before the  
 39238 beginning of the edit buffer, display the first screen of the edit buffer.

39239 Otherwise, display a screen where the last line of the display shall contain an unspecified  
 39240 portion of (current first line -1). If this calculation results in a line before the beginning of  
 39241 the edit buffer, it shall be an error.

39242 *Current line:* If *line* and the ' ^ ' character were specified:

39243 1. If the first screen was displayed as a result of the command attempting to display lines  
 39244 before the beginning of the edit buffer: if the first screen was already displayed,  
 39245 unchanged; otherwise, set to (current first line -1).

39246 2. Otherwise, set to the last line of the display.

39247 If *line* and the ' + ' character were specified, set to the first line of the display.39248 Otherwise, if *line* was specified, set to *line*.

39249 Otherwise, unchanged.

39250 *Current column*: Set to non-<blank>.

## 39251 **Exit**

39252 *Synopsis*:      ZZ

39253 This command shall be equivalent to the *ex* **xit** command with no addresses, trailing **!**, or  
39254 filename (see the *ex* **xit** command).

## 39255 **Input Mode Commands in vi**

39256 In text input mode, the current line shall consist of zero or more of the following categories, plus  
39257 the terminating <newline>:

- 39258 1. Characters preceding the text input entry point

39259 Characters in this category shall not be modified during text input mode.

- 39260 2. **autoindent** characters

39261 **autoindent** characters shall be automatically inserted into each line that is created in text  
39262 input mode, either as a result of entering a <newline> or <carriage-return> while in text  
39263 input mode, or as an effect of the command itself; for example, **O** or **o** (see the *ex*  
39264 **autoindent** command), as if entered by the user.

39265 It shall be possible to erase **autoindent** characters with the <control>-D command; it is  
39266 unspecified whether they can be erased by <control>-H, <control>-U, and <control>-W  
39267 characters. Erasing any **autoindent** character turns the glyph into erase-columns and  
39268 deletes the character from the edit buffer, but does not change its representation on the  
39269 screen.

- 39270 3. Text input characters

39271 Text input characters are the characters entered by the user. Erasing any text input  
39272 character turns the glyph into erase-columns and deletes the character from the edit buffer,  
39273 but does not change its representation on the screen.

39274 Each text input character entered by the user (that does not have a special meaning) shall  
39275 be treated as follows:

- 39276 a. The text input character shall be appended to the last character in the edit buffer  
39277 from the first, second, or third categories.
- 39278 b. If there are no erase-columns on the screen, the text input command was the **R**  
39279 command, and characters in the fifth category from the original line follow the  
39280 cursor, the next such character shall be deleted from the edit buffer. If the **slowopen**  
39281 edit option is not set, the corresponding glyph on the screen shall become erase-  
39282 columns.
- 39283 c. If there are erase-columns on the screen, as many columns as they occupy, or as are  
39284 necessary, shall be overwritten to display the text input character. (If only part of a  
39285 multi-column glyph is overwritten, the remainder shall be left on the screen, and  
39286 continue to be treated as erase-columns; it is unspecified whether the remainder of  
39287 the glyph is modified in any way.)
- 39288 d. If additional display line columns are needed to display the text input character:
  - 39289 1. If the **slowopen** edit option is set, the text input characters shall be displayed  
39290 on subsequent display line columns, overwriting any characters displayed in

39291                   those columns.

39292                   2. Otherwise, any characters currently displayed on or after the column on the  
39293                   display line where the text input character is to be displayed shall be pushed  
39294                   ahead the number of display line columns necessary to display the rest of the  
39295                   text input character.

39296           4. Erase-columns

39297                   Erase-columns are not logically part of the edit buffer, appearing only on the screen, and  
39298                   may be overwritten on the screen by subsequent text input characters. When text input  
39299                   mode ends, all erase-columns shall no longer appear on the screen.

39300                   Erase-columns are initially the region of text specified by the **c** command (see **Change** (on  
39301                   page 1009)); however, erasing **autoindent** or text input characters causes the glyphs of the  
39302                   erased characters to be treated as erase-columns.

39303           5. Characters following the text region for the **c** command, or the text input entry point for all  
39304                   other commands

39305                   Characters in this category shall not be modified during text input mode, except as  
39306                   specified in category 3.b. for the **R** text input command, or as <blank>s deleted when a  
39307                   <newline> or <carriage-return> is entered.

39308                   It is unspecified whether it is an error to attempt to erase past the beginning of a line that was  
39309                   created by the entry of a <newline> or <carriage-return> during text input mode. If it is not an  
39310                   error, the editor shall behave as if the erasing character was entered immediately after the last  
39311                   text input character entered on the previous line, and all of the non-<newline>s on the current  
39312                   line shall be treated as erase-columns.

39313                   When text input mode is entered, or after a text input mode character is entered (except as  
39314                   specified for the special characters below), the cursor shall be positioned as follows:

- 39315                   1. On the first column that displays any part of the first erase-column, if one exists
- 39316                   2. Otherwise, if the **slowopen** edit option is set, on the first display line column after the last  
39317                   character in the first, second, or third categories, if one exists
- 39318                   3. Otherwise, the first column that displays any part of the first character in the fifth category,  
39319                   if one exists
- 39320                   4. Otherwise, the display line column after the last character in the first, second, or third  
39321                   categories, if one exists
- 39322                   5. Otherwise, on column position 1

39323                   The characters that are updated on the screen during text input mode are unspecified, other than  
39324                   that the last text input character shall always be updated, and, if the **slowopen** edit option is not  
39325                   set, the current cursor character shall always be updated.

39326                   The following specifications are for command characters entered during text input mode.

39327       **NUL**

39328       *Synopsis:*       NUL

39329       If the first character of the text input is a NUL, the most recently input text shall be input as if  
 39330       entered by the user, and then text input mode shall be exited. The text shall be input literally;  
 39331       that is, characters are neither macro or abbreviation expanded, nor are any characters interpreted  
 39332       in any special manner. It is unspecified whether implementations shall support more than 256  
 39333       bytes of remembered input text.

39334       **<control>-D**

39335       *Synopsis:*       <control>-D

39336       The <control>-D character shall have no special meaning when in text input mode for a line-  
 39337       oriented command (see **Command Descriptions in vi** (on page 988)).

39338       This command need not be supported on block-mode terminals.

39339       If the cursor does not follow an **autoindent** character, or an **autoindent** character and a '0' or  
 39340       '^' character:

- 39341           1. If the cursor is in column position 1, the <control>-D character shall be discarded and no  
 39342           further action taken.
- 39343           2. Otherwise, the <control>-D character shall have no special meaning.

39344       If the last input character was a '0', the cursor shall be moved to column position 1.

39345       Otherwise, if the last input character was a '^', the cursor shall be moved to column position 1.  
 39346       In addition, the **autoindent** level for the next input line shall be derived from the same line from  
 39347       which the **autoindent** level for the current input line was derived.

39348       Otherwise, the cursor shall be moved back to the column after the previous shiftwidth (see the  
 39349       ex **shiftwidth** command) boundary.

39350       All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
 39351       cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
 39352       page 1022).

39353       *Current line:* Unchanged.

39354       *Current column:* Set to 1 if the <control>-D was preceded by a '^' or '0'; otherwise, set to  
 39355       (column - 1) - ((column - 2) % **shiftwidth**).

39356       **<control>-H**

39357       *Synopsis:*       <control>-H

39358       If in text input mode for a line-oriented command, and there are no characters to erase, text  
 39359       input mode shall be terminated, no further action shall be done for this command, and the  
 39360       current line and column shall be unchanged.

39361       If there are characters other than **autoindent** characters that have been input on the current line  
 39362       before the cursor, the cursor shall move back one character.

39363       Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
 39364       implementation-defined whether the <control>-H command is an error or if the cursor moves  
 39365       back one **autoindent** character.

39366       Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,  
 39367       it is implementation-defined whether the <control>-H command is an error or if it is equivalent

39368 to entering <control>-H after the last input character on the previous input line.

39369 Otherwise, it shall be an error.

39370 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
 39371 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
 39372 page 1022).

39373 The current erase character (see *stty*) shall cause an equivalent action to the <control>-H  
 39374 command, unless the previously inserted character was a backslash, in which case it shall be as  
 39375 if the literal current erase character had been inserted instead of the backslash.

39376 *Current line*: Unchanged, unless previously input lines are erased, in which case it shall be set to  
 39377 line -1.

39378 *Current column*: Set to the first column that displays any portion of the character backed up  
 39379 over.

39380 **<newline>**

39381 *Synopsis*:       <newline>  
 39382               <carriage-return>  
 39383               <control>-J  
 39384               <control>-M

39385 If input was part of a line-oriented command, text input mode shall be terminated and the  
 39386 command shall continue execution with the input provided.

39387 Otherwise, terminate the current line. If there are no characters other than **autoindent** characters  
 39388 on the line, all characters on the line shall be discarded. Otherwise, it is unspecified whether the  
 39389 **autoindent** characters in the line are modified by entering these characters.

39390 Continue text input mode on a new line appended after the current line. If the **slowopen** edit  
 39391 option is set, the lines on the screen below the current line shall not be pushed down, but the  
 39392 first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the  
 39393 screen below the current line shall be pushed down.

39394 If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be  
 39395 added as a prefix to the line as described by the *ex* **autoindent** edit option.

39396 All columns after the cursor that are erase-columns (as described in **Input Mode Commands in**  
 39397 **vi** (on page 1022)) shall be discarded.

39398 If the **autoindent** edit option is set, all <blank>s immediately following the cursor shall be  
 39399 discarded.

39400 All remaining characters after the cursor shall be transferred to the new line, positioned after any  
 39401 **autoindent** characters.

39402 *Current line*: Set to current line +1.

39403 *Current column*: Set to the first column that displays any portion of the first character after the  
 39404 **autoindent** characters on the new line, if any, or the first column position after the last  
 39405 **autoindent** character, if any, or column position 1.

**<control>-T**

*Synopsis:*      <control>-T

The <control>-T character shall have no special meaning when in text input mode for a line-oriented command (see **Command Descriptions in vi** (on page 988)).

This command need not be supported on block-mode terminals.

Behave as if the user entered the minimum number of <blank>s necessary to move the cursor forward to the column position after the next **shiftwidth** (see the *ex* **shiftwidth** command) boundary.

*Current line:* Unchanged.

*Current column:* Set to  $column + \text{shiftwidth} - ((column - 1) \% \text{shiftwidth})$ .

**<control>-U**

*Synopsis:*      <control>-U

If there are characters other than **autoindent** characters that have been input on the current line before the cursor, the cursor shall move to the first character input after the **autoindent** characters.

Otherwise, if there are **autoindent** characters on the current line before the cursor, it is implementation-defined whether the <control>-U command is an error or if the cursor moves to the first column position on the line.

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined whether the <control>-U command is an error or if it is equivalent to entering <control>-U after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on page 1022).

The current *kill* character (see *stty*) shall cause an equivalent action to the <control>-U command, unless the previously inserted character was a backslash, in which case it shall be as if the literal current *kill* character had been inserted instead of the backslash.

*Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to line -1.

*Current column:* Set to the first column that displays any portion of the last character backed up over.

**<control>-V**

*Synopsis:*      <control>-V  
                 <control>-Q

Allow the entry of any subsequent character, other than <control>-J or the <newline>, as a literal character, removing any special meaning that it may have to the editor in text input mode. If a <control>-V or <control>-Q is entered before a <control>-J or <newline>, the <control>-V or <control>-Q character shall be discarded, and the <control>-J or <newline> shall behave as described in the <newline> command character during input mode.

39446 For purposes of the display only, the editor shall behave as if a '^' character was entered, and  
 39447 the cursor shall be positioned as if overwriting the '^' character. When a subsequent character  
 39448 is entered, the editor shall behave as if that character was entered instead of the original  
 39449 <control>-V or <control>-Q character.

39450 *Current line:* Unchanged.

39451 *Current column:* Unchanged.

#### 39452 <control>-W

39453 *Synopsis:* <control>-W

39454 If there are characters other than **autoindent** characters that have been input on the current line  
 39455 before the cursor, the cursor shall move back over the last word preceding the cursor (including  
 39456 any <blank>s between the end of the last word and the current cursor); the cursor shall not  
 39457 move to before the first character after the end of any **autoindent** characters.

39458 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
 39459 implementation-defined whether the <control>-W command is an error or if the cursor moves to  
 39460 the first column position on the line.

39461 Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,  
 39462 it is implementation-defined whether the <control>-W command is an error or if it is equivalent  
 39463 to entering <control>-W after the last input character on the previous input line.

39464 Otherwise, it shall be an error.

39465 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
 39466 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
 39467 page 1022).

39468 *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to  
 39469 line -1.

39470 *Current column:* Set to the first column that displays any portion of the last character backed up  
 39471 over.

#### 39472 <ESC>

39473 *Synopsis:* <ESC>

39474 If input was part of a line-oriented command:

- 39475 1. If *interrupt* was entered, text input mode shall be terminated and the editor shall return to  
 39476 command mode. The terminal shall be alerted.
- 39477 2. If <ESC> was entered, text input mode shall be terminated and the command shall  
 39478 continue execution with the input provided.

39479 Otherwise, terminate text input mode and return to command mode.

39480 Any **autoindent** characters entered on newly created lines that have no other non-<newline>s  
 39481 shall be deleted.

39482 Any leading **autoindent** and <blank>s on newly created lines shall be rewritten to be the  
 39483 minimum number of <blank>s possible.

39484 The screen shall be redisplayed as necessary to match the contents of the edit buffer.

39485 *Current line:* Unchanged.

39486 *Current column:*

- 39487 1. If there are text input characters on the current line, the column shall be set to the last  
39488 column where any portion of the last text input character is displayed.
- 39489 2. Otherwise, if a character is displayed in the current column, unchanged.
- 39490 3. Otherwise, set to column position 1.

#### 39491 **EXIT STATUS**

39492 The following exit values shall be returned:

- 39493 0 Successful completion.
- 39494 >0 An error occurred.

#### 39495 **CONSEQUENCES OF ERRORS**

39496 When any error is encountered and the standard input is not a terminal device file, *vi* shall not  
39497 write the file or return to command or text input mode, and shall terminate with a non-zero exit  
39498 status.

39499 Otherwise, when an unrecoverable error is encountered it shall be equivalent to a SIGHUP  
39500 asynchronous event.

39501 Otherwise, when an error is encountered, the editor shall behave as specified in **Command**  
39502 **Descriptions in vi** (on page 988).

#### 39503 **APPLICATION USAGE**

39504 None.

#### 39505 **EXAMPLES**

39506 None.

#### 39507 **RATIONALE**

39508 See the RATIONALE for *ex* for more information on *vi*. Major portions of the *vi* utility  
39509 specification point to *ex* to avoid inadvertent divergence. While *ex* and *vi* have historically been  
39510 implemented as a single utility, this is not required by IEEE Std 1003.1-2001.

39511 It is recognized that portions of *vi* would be difficult, if not impossible, to implement  
39512 satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing,  
39513 thus it is not a mandatory requirement that such features should work on all terminals. It is the  
39514 intention, however, that a *vi* implementation should provide the full set of capabilities on all  
39515 terminals capable of supporting them.

39516 Historically, *vi* exited immediately if the standard input was not a terminal. IEEE Std 1003.1-2001  
39517 permits, but does not require, this behavior. An end-of-file condition is not equivalent to an  
39518 end-of-file character. A common end-of-file character, <control>-D, is historically a *vi* command.

39519 The text in the STDOUT section reflects the usage of the verb *display* in this section; some  
39520 implementations of *vi* use standard output to write to the terminal, but IEEE Std 1003.1-2001  
39521 does not require that to be the case.

39522 Historically, implementations reverted to open mode if the terminal was incapable of  
39523 supporting full visual mode. IEEE Std 1003.1-2001 requires this behavior. Historically, the open  
39524 mode of *vi* behaved roughly equivalently to the visual mode, with the exception that only a  
39525 single line from the edit buffer (one “buffer line”) was kept current at any time. This line was  
39526 normally displayed on the next-to-last line of a terminal with cursor addressing (and the last line  
39527 performed its normal visual functions for line-oriented commands and messages). In addition,  
39528 some few commands behaved differently in open mode than in visual mode.  
39529 IEEE Std 1003.1-2001 requires conformance to historical practice.



Historically, *ex* and *vi* implementations have expected text to proceed in the usual European/Latin order of left to right, top to bottom. There is no requirement in IEEE Std 1003.1-2001 that this be the case. The specification was deliberately written using words like “before”, “after”, “first”, and “last” in order to permit implementations to support the natural text order of the language.

Historically, lines past the end of the edit buffer were marked with single tilde ('~') characters; that is, if the one-based display was 20 lines in length, and the last line of the file was on line one, then lines 2-20 would contain only a single '~' character.

Historically, the *vi* editor attempted to display only complete lines at the bottom of the screen (it did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the bottom of the screen, the screen lines where the line would have been displayed were displayed as single '@' characters, instead of displaying part of the line. IEEE Std 1003.1-2001 permits, but does not require, this behavior. Implementations are encouraged to attempt always to display a complete line at the bottom of the screen when doing scrolling or screen positioning by buffer lines.

Historically, lines marked with '@' were also used to minimize output to dumb terminals over slow lines; that is, changes local to the cursor were updated, but changes to lines on the screen that were not close to the cursor were simply marked with an '@' sign instead of being updated to match the current text. IEEE Std 1003.1-2001 permits, but does not require this feature because it is used ever less frequently as terminals become smarter and connections are faster.

### Initialization in *ex* and *vi*

Historically, *vi* always had a line in the edit buffer, even if the edit buffer was “empty”. For example:

1. The *ex* command = executed from visual mode wrote “1” when the buffer was empty.
2. Writes from visual mode of an empty edit buffer wrote files of a single character (<newline>), while writes from *ex* mode of an empty edit buffer wrote empty files.
3. Put and read commands into an empty edit buffer left an empty line at the top of the edit buffer.

For consistency, IEEE Std 1003.1-2001 does not permit any of these behaviors.

Historically, *vi* did not always return the terminal to its original modes; for example, ICRNL was modified if it was not originally set. IEEE Std 1003.1-2001 does not permit this behavior.

### Command Descriptions in *vi*

Motion commands are among the most complicated aspects of *vi* to describe. With some exceptions, the text region and buffer type effect of a motion command on a *vi* command are described on a case-by-case basis. The descriptions of text regions in IEEE Std 1003.1-2001 are not intended to imply direction; that is, an inclusive region from line *n* to line *n*+5 is identical to a region from line *n*+5 to line *n*. This is of more than academic interest—movements to marks can be in either direction, and, if the **wrapscan** option is set, so can movements to search points. Historically, lines are always stored into buffers in text order; that is, from the start of the edit buffer to the end. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, command counts were applied to any associated motion, and were multiplicative to any supplied motion count. For example, **2cw** is the same as **c2w**, and **2c3w** is the same as **c6w**. IEEE Std 1003.1-2001 requires this behavior. Historically, *vi* commands that used bigwords, words, paragraphs, and sentences as objects treated groups of empty lines, or lines that contained only <blank>s, inconsistently. Some commands treated them as a single entity, while

others treated each line separately. For example, the **w**, **W**, and **B** commands treated groups of empty lines as individual words; that is, the command would move the cursor to each new empty line. The **e** and **E** commands treated groups of empty lines as a single word; that is, the first use would move past the group of lines. The **b** command would just beep at the user, or if done from the start of the line as a motion command, fail in unexpected ways. If the lines contained only (or ended with) `<blank>s`, the **w** and **W** commands would just beep at the user, the **E** and **e** commands would treat the group as a single word, and the **B** and **b** commands would treat the lines as individual words. For consistency and simplicity of specification, IEEE Std 1003.1-2001 requires that all *vi* commands treat groups of empty or blank lines as a single entity, and that movement through lines ending with `<blank>s` be consistent with other movements.

Historically, *vi* documentation indicated that any number of double quotes were skipped after punctuation marks at sentence boundaries; however, implementations only skipped single quotes. IEEE Std 1003.1-2001 requires both to be skipped.

Historically, the first and last characters in the edit buffer were word boundaries. This historical practice is required by IEEE Std 1003.1-2001.

Historically, *vi* attempted to update the minimum number of columns on the screen possible, which could lead to misleading information being displayed. IEEE Std 1003.1-2001 makes no requirements other than that the current character being entered is displayed correctly, leaving all other decisions in this area up to the implementation.

Historically, lines were arbitrarily folded between columns of any characters that required multiple column positions on the screen, with the exception of tabs, which terminated at the right-hand margin. IEEE Std 1003.1-2001 permits the former and requires the latter. Implementations that do not arbitrarily break lines between columns of characters that occupy multiple column positions should not permit the cursor to rest on a column that does not contain any part of a character.

The historical *vi* had a problem in that all movements were by buffer lines, not by display or screen lines. This is often the right thing to do; for example, single line movements, such as **j** or **k**, should work on buffer lines. Commands like **dj**, or **j.**, where **.** is a change command, only make sense for buffer lines. It is not, however, the right thing to do for screen motion or scrolling commands like `<control>-D`, `<control>-F`, and **H**. If the window is fairly small, using buffer lines in these cases can result in completely random motion; for example, **1<control>-D** can result in a completely changed screen, without any overlap. This is clearly not what the user wanted. The problem is even worse in the case of the **H**, **L**, and **M** commands—as they position the cursor at the first non-`<blank>` of the line, they may all refer to the same location in large lines, and will result in no movement at all.

In addition, if the line is larger than the screen, using buffer lines can make it impossible to display parts of the line—there are not any commands that do not display the beginning of the line in historical *vi*, and if both the beginning and end of the line cannot be on the screen at the same time, the user suffers. Finally, the page and half-page scrolling commands historically moved to the first non-`<blank>` in the new line. If the line is approximately the same size as the screen, this is inadequate because the cursor before and after a `<control>-D` command will refer to the same location on the screen.

Implementations of *ex* and *vi* exist that do not have these problems because the relevant commands (`<control>-B`, `<control>-D`, `<control>-F`, `<control>-U`, `<control>-Y`, `<control>-E`, **H**, **L**, and **M**) operate on display (screen) lines, not (edit) buffer lines.

IEEE Std 1003.1-2001 does not permit this behavior by default because the standard developers believed that users would find it too confusing. However, historical practice has been relaxed.

For example, *ex* and *vi* historically attempted, albeit sometimes unsuccessfully, to never put part of a line on the last lines of a screen; for example, if a line would not fit in its entirety, no part of the line was displayed, and the screen lines corresponding to the line contained single '@' characters. This behavior is permitted, but not required by IEEE Std 1003.1-2001, so that it is possible for implementations to support long lines in small screens more reasonably without changing the commands to be oriented to the display (instead of oriented to the buffer). IEEE Std 1003.1-2001 also permits implementations to refuse to edit any edit buffer containing a line that will not fit on the screen in its entirety.

The display area (for example, the value of the **window** edit option) has historically been “grown”, or expanded, to display new text when local movements are done in displays where the number of lines displayed is less than the maximum possible. Expansion has historically been the first choice, when the target line is less than the maximum possible expansion value away. Scrolling has historically been the next choice, done when the target line is less than half a display away, and otherwise, the screen was redrawn. There were exceptions, however, in that *ex* commands generally always caused the screen to be redrawn. IEEE Std 1003.1-2001 does not specify a standard behavior because there may be external issues, such as connection speed, the number of characters necessary to redraw as opposed to scroll, or terminal capabilities that implementations will have to accommodate.

The current line in IEEE Std 1003.1-2001 maps one-to-one to a buffer line in the file. The current column does not. There are two different column values that are described by IEEE Std 1003.1-2001. The first is the current column value as set by many of the *vi* commands. This value is remembered for the lifetime of the editor. The second column value is the actual position on the screen where the cursor rests. The two are not always the same. For example, when the cursor is backed by a multi-column character, the actual cursor position on the screen has historically been the last column of the character in command mode, and the first column of the character in input mode.

Commands that set the current line, but that do not set the current cursor value (for example, **j** and **k**) attempt to get as close as possible to the remembered column position, so that the cursor tends to restrict itself to a vertical column as the user moves around in the edit buffer. IEEE Std 1003.1-2001 requires conformance to historical practice, requiring that the display location of the cursor on the display line be adjusted from the current column value as necessary to support this historical behavior.

Historically, only a single line (and for some terminals, a single line minus 1 column) of characters could be entered by the user for the line-oriented commands; that is, **:**, **!**, **/**, or **?**. IEEE Std 1003.1-2001 permits, but does not require, this limitation.

Historically, “soft” errors in *vi* caused the terminal to be alerted, but no error message was displayed. As a general rule, no error message was displayed for errors in command execution in *vi*, when the error resulted from the user attempting an invalid or impossible action, or when a searched-for object was not found. Examples of soft errors included **h** at the left margin, **<control>-B** or **[[** at the beginning of the file, **2G** at the end of the file, and so on. In addition, errors such as **%**, **[[**, **]**, **)**, **N**, **n**, **f**, **F**, **t**, and **T** failing to find the searched-for object were soft as well. Less consistently, **/** and **?** displayed an error message if the pattern was not found, **/**, **?**, **N**, and **n** displayed an error message if no previous regular expression had been specified, and **;** did not display an error message if no previous **f**, **F**, **t**, or **T** command had occurred. Also, behavior in this area might reasonably be based on a runtime evaluation of the speed of a network connection. Finally, some implementations have provided error messages for soft errors in order to assist naive users, based on the value of a verbose edit option. IEEE Std 1003.1-2001 does not list specific errors for which an error message shall be displayed. Implementations should conform to historical practice in the absence of any strong reason to diverge.

### Page Backwards

The <control>-B and <control>-F commands historically considered it an error to attempt to page past the beginning or end of the file, whereas the <control>-D and <control>-U commands simply moved to the beginning or end of the file. For consistency, IEEE Std 1003.1-2001 requires the latter behavior for all four commands. All four commands still consider it an error if the current line is at the beginning (<control>-B, <control>-U) or end (<control>-F, <control>-D) of the file. Historically, the <control>-B and <control>-F commands skip two lines in order to include overlapping lines when a single command is entered. This makes less sense in the presence of a *count*, as there will be, by definition, no overlapping lines. The actual calculation used by historical implementations of the *vi* editor for <control>-B was:

$$((\text{current first line}) - \text{count} \times (\text{window edit option})) + 2$$

and for <control>-F was:

$$((\text{current first line}) + \text{count} \times (\text{window edit option})) - 2$$

This calculation does not work well when intermixing commands with and without counts; for example, 3<control>-F is not equivalent to entering the <control>-F command three times, and is not reversible by entering the <control>-B command three times. For consistency with other *vi* commands that take counts, IEEE Std 1003.1-2001 requires a different calculation.

### Scroll Forward

The 4BSD and System V implementations of *vi* differed on the initial value used by the **scroll** command. 4BSD used:

$$((\text{window edit option}) + 1) / 2$$

while System V used the value of the **scroll** edit option. The System V version is specified by IEEE Std 1003.1-2001 because the standard developers believed that it was more intuitive and permitted the user a method of setting the scroll value initially without also setting the number of lines that are displayed.

### Scroll Forward by Line

Historically, the <control>-E and <control>-Y commands considered it an error if the last and first lines, respectively, were already on the screen. IEEE Std 1003.1-2001 requires conformance to historical practice. Historically, the <control>-E and <control>-Y commands had no effect in open mode. For simplicity and consistency of specification, IEEE Std 1003.1-2001 requires that they behave as usual, albeit with a single line screen.

### Clear and Redisplay

The historical <control>-L command refreshed the screen exactly as it was supposed to be currently displayed, replacing any '@' characters for lines that had been deleted but not updated on the screen with refreshed '@' characters. The intent of the <control>-L command is to refresh when the screen has been accidentally overwritten; for example, by a **write** command from another user, or modem noise.

### 39709 **Redraw Screen**

39710 The historical <control>-R command redisplayed only when necessary to update lines that had  
 39711 been deleted but not updated on the screen and that were flagged with '@' characters. There is  
 39712 no requirement that the screen be in any way refreshed if no lines of this form are currently  
 39713 displayed. IEEE Std 1003.1-2001 permits implementations to extend this command to refresh  
 39714 lines on the screen flagged with '@' characters because they are too long to be displayed in the  
 39715 current framework; however, the current line and column need not be modified.

### 39716 **Search for tagstring**

39717 Historically, the first non-<blank> at or after the cursor was the first character, and all  
 39718 subsequent characters that were word characters, up to the end of the line, were included. For  
 39719 example, with the cursor on the leading space or on the '#' character in the text "#bar@", the  
 39720 tag was "#bar". On the character 'b' it was "bar", and on the 'a' it was "ar".  
 39721 IEEE Std 1003.1-2001 requires this behavior.

### 39722 **Replace Text with Results from Shell Command**

39723 Historically, the <, >, and ! commands considered most cursor motions other than line-oriented  
 39724 motions an error; for example, the command >/foo<CR> succeeded, while the command >I  
 39725 failed, even though the text region described by the two commands might be identical. For  
 39726 consistency, all three commands only consider entire lines and not partial lines, and the region is  
 39727 defined as any line that contains a character that was specified by the motion.

### 39728 **Move to Matching Character**

39729 Other matching characters have been left implementation-defined in order to allow extensions  
 39730 such as matching '<' and '>' for searching HTML, or #ifdef, #else, and #endif for searching C  
 39731 source.

### 39732 **Repeat Substitution**

39733 IEEE Std 1003.1-2001 requires that any c and g flags specified to the previous substitute  
 39734 command be ignored; however, the r flag may still apply, if supported by the implementation.

### 39735 **Return to Previous (Context or Section)**

39736 The [[, ]], (, ), {, and } commands are all affected by “section boundaries”, but in some historical  
 39737 implementations not all of the commands recognize the same section boundaries. This is a bug,  
 39738 not a feature, and a unique section-boundary algorithm was not described for each command.  
 39739 One special case that is preserved is that the sentence command moves to the end of the last line  
 39740 of the edit buffer while the other commands go to the beginning, in order to preserve the  
 39741 traditional character cut semantics of the sentence command. Historically, vi section boundaries  
 39742 at the beginning and end of the edit buffer were the first non-<blank> on the first and last lines  
 39743 of the edit buffer if one exists; otherwise, the last character of the first and last lines of the edit  
 39744 buffer if one exists. To increase consistency with other section locations, this has been simplified  
 39745 by IEEE Std 1003.1-2001 to the first character of the first and last lines of the edit buffer, or the  
 39746 first and the last lines of the edit buffer if they are empty.

39747 Sentence boundaries were problematic in the historical vi. They were not only the boundaries as  
 39748 defined for the section and paragraph commands, but they were the first non-<blank> that  
 39749 occurred after those boundaries, as well. Historically, the vi section commands were  
 39750 documented as taking an optional window size as a count preceding the command. This was not  
 39751 implemented in historical versions, so IEEE Std 1003.1-2001 requires that the count repeat the  
 39752 command, for consistency with other vi commands.

**Repeat**

Historically, mapped commands other than text input commands could not be repeated using the **period** command. IEEE Std 1003.1-2001 requires conformance to historical practice.

The restrictions on the interpretation of special characters (for example, <control>-H) in the repetition of text input mode commands is intended to match historical practice. For example, given the input sequence:

```
iab<control>-H<control>-H<control>-Hdef<escape>
```

the user should be informed of an error when the sequence is first entered, but not during a command repetition. The character <control>-T is specifically exempted from this restriction. Historical implementations of *vi* ignored <control>-T characters that were input in the original command during command repetition. IEEE Std 1003.1-2001 prohibits this behavior.

**Find Regular Expression**

Historically, commands did not affect the line searched to or from if the motion command was a search (*/*, *?*, **N**, **n**) and the final position was the start/end of the line. There were some special cases and *vi* was not consistent. IEEE Std 1003.1-2001 does not permit this behavior, for consistency. Historical implementations permitted but were unable to handle searches as motion commands that wrapped (that is, due to the edit option **wrapsan**) to the original location. IEEE Std 1003.1-2001 requires that this behavior be treated as an error.

Historically, the syntax *"/RE/0"* was used to force the command to cut text in line mode. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, in open mode, a **z** specified to a search command redisplayed the current line instead of displaying the current screen with the current line highlighted. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

Historically, trailing **z** commands were permitted and ignored if entered as part of a search used as a motion command. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

**Execute an ex Command**

Historically, *vi* implementations restricted the commands that could be entered on the colon command line (for example, **append** and **change**), and some other commands were known to cause them to fail catastrophically. For consistency, IEEE Std 1003.1-2001 does not permit these restrictions. When executing an *ex* command by entering *:*, it is not possible to enter a <newline> as part of the command because it is considered the end of the command. A different approach is to enter *ex* command mode by using the *vi* **Q** command (and later resuming visual mode with the *ex* **vi** command). In *ex* command mode, the single-line limitation does not exist. So, for example, the following is valid:

```
Q
s/break here/break\
here/
vi
```

IEEE Std 1003.1-2001 requires that, if the *ex* command overwrites any part of the screen that would be erased by a refresh, *vi* pauses for a character from the user. Historically, this character could be any character; for example, a character input by the user before the message appeared, or even a mapped character. This is probably a bug, but implementations that have tried to be more rigorous by requiring that the user enter a specific character, or that the user enter a character after the message was displayed, have been forced by user indignation back into

39798 historical behavior. IEEE Std 1003.1-2001 requires conformance to historical practice.

#### 39799 **Shift Left (Right)**

39800 Refer to the Rationale for the **!** and **/** commands. Historically, the **<** and **>** commands sometimes  
39801 moved the cursor to the first non-**<blank>** (for example if the command was repeated or with **\_**  
39802 as the motion command), and sometimes left it unchanged. IEEE Std 1003.1-2001 does not  
39803 permit this inconsistency, requiring instead that the cursor always move to the first non-  
39804 **<blank>**. Historically, the **<** and **>** commands did not support buffer arguments, although some  
39805 implementations allow the specification of an optional buffer. This behavior is neither required  
39806 nor disallowed by IEEE Std 1003.1-2001.

#### 39807 **Execute**

39808 Historically, buffers could execute other buffers, and loops, infinite and otherwise, were  
39809 possible. IEEE Std 1003.1-2001 requires conformance to historical practice. The *\*buffer* syntax of  
39810 **ex** is not required in **vi**, because it is not historical practice and has been used in some **vi**  
39811 implementations to support additional scripting languages.

#### 39812 **Reverse Case**

39813 Historically, the **~** command ignored any associated *count*, and acted only on the characters in  
39814 the current line. For consistency with other **vi** commands, IEEE Std 1003.1-2001 requires that an  
39815 associated *count* act on the next *count* characters, and that the command move to subsequent  
39816 lines if warranted by *count*, to make it possible to modify large pieces of text in a reasonably  
39817 efficient manner. There exist **vi** implementations that optionally require an associated motion  
39818 command for the **~** command. Implementations supporting this functionality are encouraged to  
39819 base it on the **tildedop** edit option and handle the text regions and cursor positioning identically  
39820 to the **yank** command.

#### 39821 **Append**

39822 Historically, *counts* specified to the **A**, **a**, **I**, and **i** commands repeated the input of the first line  
39823 *count* times, and did not repeat the subsequent lines of the input text. IEEE Std 1003.1-2001  
39824 requires that the entire text input be repeated *count* times.

#### 39825 **Move Backward to Preceding Word**

39826 Historically, **vi** became confused if word commands were used as motion commands in empty  
39827 files. IEEE Std 1003.1-2001 requires that this be an error. Historical implementations of **vi** had a  
39828 large number of bugs in the word movement commands, and they varied greatly in behavior in  
39829 the presence of empty lines, “words” made up of a single character, and lines containing only  
39830 **<blank>**s. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit  
39831 this behavior.

#### 39832 **Change to End-of-Line**

39833 Some historical implementations of the **C** command did not behave as described by  
39834 IEEE Std 1003.1-2001 when the **\$** key was remapped because they were implemented by pushing  
39835 the **\$** key onto the input queue and reprocessing it. IEEE Std 1003.1-2001 does not permit this  
39836 behavior. Historically, the **C**, **S**, and **s** commands did not copy replaced text into the numeric  
39837 buffers. For consistency and simplicity of specification, IEEE Std 1003.1-2001 requires that they  
39838 behave like their respective **c** commands in all respects.

**Delete**

Historically, lines in open mode that were deleted were scrolled up, and an @ glyph written over the beginning of the line. In the case of terminals that are incapable of the necessary cursor motions, the editor erased the deleted line from the screen. IEEE Std 1003.1-2001 requires conformance to historical practice; that is, if the terminal cannot display the '@' character, the line cannot remain on the screen.

**Delete to End-of-Line**

Some historical implementations of the **D** command did not behave as described by IEEE Std 1003.1-2001 when the **\$** key was remapped because they were implemented by pushing the **\$** key onto the input queue and reprocessing it. IEEE Std 1003.1-2001 does not permit this behavior.

**Join**

An historical oddity of *vi* is that the commands **J**, **1J**, and **2J** are all equivalent. IEEE Std 1003.1-2001 requires conformance to historical practice. The *vi* **J** command is specified in terms of the *ex* **join** command with an *ex* command *count* value. The address correction for a *count* that is past the end of the edit buffer is necessary for historical compatibility for both *ex* and *vi*.

**Mark Position**

Historical practice is that only lowercase letters, plus '' and ''', could be used to mark a cursor position. IEEE Std 1003.1-2001 requires conformance to historical practice, but encourages implementations to support other characters as marks as well.

**Repeat Regular Expression Find (Forward and Reverse)**

Historically, the **N** and **n** commands could not be used as motion components for the **c** command. With the exception of the **cN** command, which worked if the search crossed a line boundary, the text region would be discarded, and the user would not be in text input mode. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

**Insert Empty Line (Below and Above)**

Historically, counts to the **O** and **o** commands were used as the number of physical lines to open, if the terminal was dumb and the **slowopen** option was not set. This was intended to minimize traffic over slow connections and repainting for dumb terminals. IEEE Std 1003.1-2001 does not permit this behavior, requiring that a *count* to the open command behave as for other text input commands. This change to historical practice was made for consistency, and because a superset of the functionality is provided by the **slowopen** edit option.

**Put from Buffer (Following and Before)**

Historically, *counts* to the **p** and **P** commands were ignored if the buffer was a line mode buffer, but were (mostly) implemented as described in IEEE Std 1003.1-2001 if the buffer was a character mode buffer. Because implementations exist that do not have this limitation, and because pasting lines multiple times is generally useful, IEEE Std 1003.1-2001 requires that *count* be supported for all **p** and **P** commands.

Historical implementations of *vi* were widely known to have major problems in the **p** and **P** commands, particularly when unusual regions of text were copied into the edit buffer. The standard developers viewed these as bugs, and they are not permitted for consistency and



39881 simplicity of specification.

39882 Historically, a **P** or **p** command (or an **ex put** command executed from open or visual mode)  
39883 executed in an empty file, left an empty line as the first line of the file. For consistency and  
39884 simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

#### 39885 **Replace Character**

39886 Historically, the **r** command did not correctly handle the *erase* and *word erase* characters as  
39887 arguments, nor did it handle an associated *count* greater than 1 with a <carriage-return>  
39888 argument, for which it replaced *count* characters with a single <newline>. IEEE Std 1003.1-2001  
39889 does not permit these inconsistencies.

39890 Historically, the **r** command permitted the <control>-V escaping of entered characters, such as  
39891 <ESC> and the <carriage-return>; however, it required two leading <control>-V characters  
39892 instead of one. IEEE Std 1003.1-2001 requires that this be changed for consistency with the other  
39893 text input commands of *vi*.

39894 Historically, it is an error to enter the **r** command if there are less than *count* characters at or after  
39895 the cursor in the line. While a reasonable and unambiguous extension would be to permit the **r**  
39896 command on empty lines, it would require that too large a *count* be adjusted to match the  
39897 number of characters at or after the cursor for consistency, which is sufficiently different from  
39898 historical practice to be avoided. IEEE Std 1003.1-2001 requires conformance to historical  
39899 practice.

#### 39900 **Replace Characters**

39901 Historically, if there were **autoindent** characters in the line on which the **R** command was run,  
39902 and **autoindent** was set, the first <newline> would be properly indented and no characters  
39903 would be replaced by the <newline>. Each additional <newline> would replace *n* characters,  
39904 where *n* was the number of characters that were needed to indent the rest of the line to the  
39905 proper indentation level. This behavior is a bug and is not permitted by IEEE Std 1003.1-2001.

#### 39906 **Undo**

39907 Historical practice for cursor positioning after undoing commands was mixed. In most cases,  
39908 when undoing commands that affected a single line, the cursor was moved to the start of added  
39909 or changed text, or immediately after deleted text. However, if the user had moved from the line  
39910 being changed, the column was either set to the first non-<blank>, returned to the origin of the  
39911 command, or remained unchanged. When undoing commands that affected multiple lines or  
39912 entire lines, the cursor was moved to the first character in the first line restored. As an example  
39913 of how inconsistent this was, a search, followed by an **o** text input command, followed by an  
39914 **undo** would return the cursor to the location where the **o** command was entered, but a **cw**  
39915 command followed by an **o** command followed by an **undo** would return the cursor to the first  
39916 non-<blank> of the line. IEEE Std 1003.1-2001 requires the most useful of these behaviors, and  
39917 discards the least useful, in the interest of consistency and simplicity of specification.

**Yank**

Historically, the **yank** command did not move to the end of the motion if the motion was in the forward direction. It moved to the end of the motion if the motion was in the backward direction, except for the **\_** command, or for the **G** and **'** commands when the end of the motion was on the current line. This was further complicated by the fact that for a number of motion commands, the **yank** command moved the cursor but did not update the screen; for example, a subsequent command would move the cursor from the end of the motion, even though the cursor on the screen had not reflected the cursor movement for the **yank** command. IEEE Std 1003.1-2001 requires that all **yank** commands associated with backward motions move the cursor to the end of the motion for consistency, and specifically, to make **'** commands as motions consistent with search patterns as motions.

**Yank Current Line**

Some historical implementations of the **Y** command did not behave as described by IEEE Std 1003.1-2001 when the **'\_'** key was remapped because they were implemented by pushing the **'\_'** key onto the input queue and reprocessing it. IEEE Std 1003.1-2001 does not permit this behavior.

**Redraw Window**

Historically, the **z** command always redrew the screen. This is permitted but not required by IEEE Std 1003.1-2001, because of the frequent use of the **z** command in macros such as **map n nz.** for screen positioning, instead of its use to change the screen size. The standard developers believed that expanding or scrolling the screen offered a better interface for users. The ability to redraw the screen is preserved if the optional new window size is specified, and in the **<control>-L** and **<control>-R** commands.

The semantics of **z^** are confusing at best. Historical practice is that the screen before the screen that ended with the specified line is displayed. IEEE Std 1003.1-2001 requires conformance to historical practice.

Historically, the **z** command would not display a partial line at the top or bottom of the screen. If the partial line would normally have been displayed at the bottom of the screen, the command worked, but the partial line was replaced with **'@'** characters. If the partial line would normally have been displayed at the top of the screen, the command would fail. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

Historically, the **z** command with a line specification of 1 ignored the command. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

Historically, the **z** command did not set the cursor column to the first non-**<blank>** for the character if the first screen was to be displayed, and was already displayed. For consistency and simplicity of specification, IEEE Std 1003.1-2001 does not permit this behavior.

**Input Mode Commands in vi**

Historical implementations of **vi** did not permit the user to erase more than a single line of input, or to use normal erase characters such as *line erase*, *worderase*, and *erase* to erase **autoindent** characters. As there exist implementations of **vi** that do not have these limitations, both behaviors are permitted, but only historical practice is required. In the case of these extensions, **vi** is required to pause at the **autoindent** and previous line boundaries.

Historical implementations of **vi** updated only the portion of the screen where the current cursor character was displayed. For example, consider the **vi** input keystrokes:

39962 iabcd<escape>0C<tab>

39963 Historically, the <tab> would overwrite the characters "abcd" when it was displayed. Other  
 39964 implementations replace only the 'a' character with the <tab>, and then push the rest of the  
 39965 characters ahead of the cursor. Both implementations have problems. The historical  
 39966 implementation is probably visually nicer for the above example; however, for the keystrokes:

39967 iabcd<ESC>0R<tab><ESC>

39968 the historical implementation results in the string "bcd" disappearing and then magically  
 39969 reappearing when the <ESC> character is entered. IEEE Std 1003.1-2001 requires the former  
 39970 behavior when overwriting erase-columns—that is, overwriting characters that are no longer  
 39971 logically part of the edit buffer—and the latter behavior otherwise.

39972 Historical implementations of *vi* discarded the <control>-D and <control>-T characters when  
 39973 they were entered at places where their command functionality was not appropriate.  
 39974 IEEE Std 1003.1-2001 requires that the <control>-T functionality always be available, and that  
 39975 <control>-D be treated as any other key when not operating on **autoindent** characters.

## 39976 NUL

39977 Some historical implementations of *vi* limited the number of characters entered using the NUL  
 39978 input character to 256 bytes. IEEE Std 1003.1-2001 permits this limitation; however,  
 39979 implementations are encouraged to remove this limit.

## 39980 <control>-D

39981 See also Rationale for the input mode command <newline>. The hidden assumptions in the  
 39982 <control>-D command (and in the *vi* **autoindent** specification in general) is that <space>s take  
 39983 up a single column on the screen and that <tab>s are comprised of an integral number of  
 39984 <space>s.

## 39985 <newline>

39986 Implementations are permitted to rewrite **autoindent** characters in the line when <newline>,  
 39987 <carriage-return>, <control>-D, and <control>-T are entered, or when the **shift** commands are  
 39988 used, because historical implementations have both done so and found it necessary to do so. For  
 39989 example, a <control>-D when the cursor is preceded by a single <tab>, with **tabstop** set to 8, and  
 39990 **shiftwidth** set to 3, will result in the <tab> being replaced by several <space>s.

## 39991 <control>-T

39992 See also the Rationale for the input mode command <newline>. Historically, <control>-T only  
 39993 worked if no non-<blank>s had yet been input in the current input line. In addition, the  
 39994 characters inserted by <control>-T were treated as **autoindent** characters, and could not be  
 39995 erased using normal user erase characters. Because implementations exist that do not have  
 39996 these limitations, and as moving to a column boundary is generally useful, IEEE Std 1003.1-2001  
 39997 requires that both limitations be removed.

|       |                                                                                                                     |
|-------|---------------------------------------------------------------------------------------------------------------------|
| 39998 | <b>&lt;control&gt;-V</b>                                                                                            |
| 39999 | Historically, <i>vi</i> used <b>^V</b> , regardless of the value of the literal-next character of the terminal.     |
| 40000 | IEEE Std 1003.1-2001 requires conformance to historical practice.                                                   |
| 40001 | The uses described for <control>-V can also be accomplished with <control>-Q, which is useful                       |
| 40002 | on terminals that use <control>-V for the down-arrow function. However, most historical                             |
| 40003 | implementations use <control>-Q for the <i>termios</i> START character, so the editor will generally                |
| 40004 | not receive the <control>-Q unless <b>stty ixon</b> mode is set to off. (In addition, some historical               |
| 40005 | implementations of <i>vi</i> explicitly set <b>ixon</b> mode to on, so it was difficult for the user to set it to   |
| 40006 | off.) Any of the command characters described in IEEE Std 1003.1-2001 can be made ineffective                       |
| 40007 | by their selection as <i>termios</i> control characters, using the <i>stty</i> utility or other methods described   |
| 40008 | in the System Interfaces volume of IEEE Std 1003.1-2001.                                                            |
| 40009 | <b>&lt;ESC&gt;</b>                                                                                                  |
| 40010 | Historically, SIGINT alerted the terminal when used to end input mode. This behavior is                             |
| 40011 | permitted, but not required, by IEEE Std 1003.1-2001.                                                               |
| 40012 | <b>FUTURE DIRECTIONS</b>                                                                                            |
| 40013 | None.                                                                                                               |
| 40014 | <b>SEE ALSO</b>                                                                                                     |
| 40015 | <i>ed</i> , <i>ex</i> , <i>stty</i>                                                                                 |
| 40016 | <b>CHANGE HISTORY</b>                                                                                               |
| 40017 | First released in Issue 2.                                                                                          |
| 40018 | <b>Issue 5</b>                                                                                                      |
| 40019 | The FUTURE DIRECTIONS section is added.                                                                             |
| 40020 | <b>Issue 6</b>                                                                                                      |
| 40021 | This utility is marked as part of the User Portability Utilities option.                                            |
| 40022 | The APPLICATION USAGE section is added.                                                                             |
| 40023 | The obsolescent SYNOPSIS is removed.                                                                                |
| 40024 | The following new requirements on POSIX implementations derive from alignment with the                              |
| 40025 | Single UNIX Specification:                                                                                          |
| 40026 | <ul style="list-style-type: none"> <li>• The <b>reindent</b> command description is added.</li> </ul>               |
| 40027 | The <i>vi</i> utility has been extensively rewritten for alignment with the IEEE P1003.2b draft                     |
| 40028 | standard.                                                                                                           |
| 40029 | IEEE PASC Interpretations 1003.2 #57, #62, #63, #64, #78, and #188 are applied.                                     |
| 40030 | IEEE PASC Interpretation 1003.2 #207 is applied, clarifying the description of the <b>R</b> command in              |
| 40031 | a manner similar to the descriptions of other text input mode commands such as <b>i</b> , <b>o</b> , and <b>O</b> . |
| 40032 | The <b>-I</b> option is removed.                                                                                    |
| 40033 | IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/41 is applied, adding [count] to the                               |
| 40034 | Synopsis for <b>[[</b> .                                                                                            |
| 40035 | IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/42 is applied, adding [count] to the                               |
| 40036 | Synopsis for <b>]]</b> .                                                                                            |

40037 **NAME**

40038           wait — await process completion

40039 **SYNOPSIS**40040           wait [*pid*...]40041 **DESCRIPTION**

40042           When an asynchronous list (see Section 2.9.3.1 (on page 50)) is started by the shell, the process ID  
 40043           of the last command in each element of the asynchronous list shall become known in the current  
 40044           shell execution environment; see Section 2.12 (on page 61).

40045           If the *wait* utility is invoked with no operands, it shall wait until all process IDs known to the  
 40046           invoking shell have terminated and exit with a zero exit status.

40047           If one or more *pid* operands are specified that represent known process IDs, the *wait* utility shall  
 40048           wait until all of them have terminated. If one or more *pid* operands are specified that represent  
 40049           unknown process IDs, *wait* shall treat them as if they were known process IDs that exited with  
 40050           exit status 127. The exit status returned by the *wait* utility shall be the exit status of the process  
 40051           requested by the last *pid* operand.

40052           The known process IDs are applicable only for invocations of *wait* in the current shell execution  
 40053           environment.

40054 **OPTIONS**

40055           None.

40056 **OPERANDS**

40057           The following operand shall be supported:

40058           *pid*           One of the following:

- 40059                           1. The unsigned decimal integer process ID of a command, for which the utility  
 40060                           is to wait for the termination.
- 40061                           2. A job control job ID (see the Base Definitions volume of IEEE Std 1003.1-2001,  
 40062                           Section 3.203, Job Control Job ID) that identifies a background process group  
 40063                           to be waited for. The job control job ID notation is applicable only for  
 40064                           invocations of *wait* in the current shell execution environment; see Section  
 40065                           2.12 (on page 61). The exit status of *wait* shall be determined by the last  
 40066                           command in the pipeline.

40067                           **Note:**   The job control job ID type of *pid* is only available on systems supporting  
 40068                           the User Portability Utilities option.

40069 **STDIN**

40070           Not used.

40071 **INPUT FILES**

40072           None.

40073 **ENVIRONMENT VARIABLES**40074           The following environment variables shall affect the execution of *wait*:

40075           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 40076                           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 40077                           Internationalization Variables for the precedence of internationalization variables  
 40078                           used to determine the values of locale categories.)

40079           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 40080                           internationalization variables.

40081 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 40082 characters (for example, single-byte as opposed to multi-byte characters in  
 40083 arguments).

40084 **LC\_MESSAGES**  
 40085 Determine the locale that should be used to affect the format and contents of  
 40086 diagnostic messages written to standard error.

40087 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

40088 **ASYNCHRONOUS EVENTS**  
 40089 Default.

40090 **STDOUT**  
 40091 Not used.

40092 **STDERR**  
 40093 The standard error shall be used only for diagnostic messages.

40094 **OUTPUT FILES**  
 40095 None.

40096 **EXTENDED DESCRIPTION**  
 40097 None.

40098 **EXIT STATUS**  
 40099 If one or more operands were specified, all of them have terminated or were not known by the  
 40100 invoking shell, and the status of the last operand specified is known, then the exit status of *wait*  
 40101 shall be the exit status information of the command indicated by the last operand specified. If  
 40102 the process terminated abnormally due to the receipt of a signal, the exit status shall be greater  
 40103 than 128 and shall be distinct from the exit status generated by other signals, but the exact value  
 40104 is unspecified. (See the *kill -l* option.) Otherwise, the *wait* utility shall exit with one of the  
 40105 following values:

40106 0 The *wait* utility was invoked with no operands and all process IDs known by the  
 40107 invoking shell have terminated.

40108 1-126 The *wait* utility detected an error.

40109 127 The command identified by the last *pid* operand specified is unknown.

40110 **CONSEQUENCES OF ERRORS**  
 40111 Default.

40112 **APPLICATION USAGE**  
 40113 On most implementations, *wait* is a shell built-in. If it is called in a subshell or separate utility  
 40114 execution environment, such as one of the following:

40115 (wait)  
 40116 nohup wait ...  
 40117 find . -exec wait ... \;

40118 it returns immediately because there are no known process IDs to wait for in those  
 40119 environments.

40120 Historical implementations of interactive shells have discarded the exit status of terminated  
 40121 background processes before each shell prompt. Therefore, the status of background processes  
 40122 was usually lost unless it terminated while *wait* was waiting for it. This could be a serious  
 40123 problem when a job that was expected to run for a long time actually terminated quickly with a  
 40124 syntax or initialization error because the exit status returned was usually zero if the requested

process ID was not found. This volume of IEEE Std 1003.1-2001 requires the implementation to keep the status of terminated jobs available until the status is requested, so that scripts like:

```

40127 j1&
40128 p1=$!
40129 j2&
40130 wait $p1
40131 echo Job 1 exited with status $?
40132 wait $!
40133 echo Job 2 exited with status $?

```

work without losing status on any of the jobs. The shell is allowed to discard the status of any process if it determines that the application cannot get the process ID for that process from the shell. It is also required to remember only {CHILD\_MAX} number of processes in this way. Since the only way to get the process ID from the shell is by using the '!' shell parameter, the shell is allowed to discard the status of an asynchronous list if "\$!" was not referenced before another asynchronous list was started. (This means that the shell only has to keep the status of the last asynchronous list started if the application did not reference "\$!". If the implementation of the shell is smart enough to determine that a reference to "\$!" was not saved anywhere that the application can retrieve it later, it can use this information to trim the list of saved information. Note also that a successful call to *wait* with no operands discards the exit status of all asynchronous lists.)

If the exit status of *wait* is greater than 128, there is no way for the application to know if the waited-for process exited with that value or was killed by a signal. Since most utilities exit with small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications just need to know that the asynchronous job failed; it does not matter whether it detected an error and failed or was killed and did not complete its job normally.

#### 40150 EXAMPLES

Although the exact value used when a process is terminated by a signal is unspecified, if it is known that a signal terminated a process, a script can still reliably determine which signal by using *kill* as shown by the following script:

```

40154 sleep 1000&
40155 pid=$!
40156 kill -kill $pid
40157 wait $pid
40158 echo $pid was terminated by a SIG$(kill -l $?) signal.

```

If the following sequence of commands is run in less than 31 seconds:

```

40160 sleep 257 | sleep 31 &
40161 jobs -l %%

```

either of the following commands returns the exit status of the second *sleep* in the pipeline:

```

40163 wait <pid of sleep 31>
40164 wait %%

```

#### 40165 RATIONALE

The description of *wait* does not refer to the *waitpid()* function from the System Interfaces volume of IEEE Std 1003.1-2001 because that would needlessly overspecify this interface. However, the wording means that *wait* is required to wait for an explicit process when it is given an argument so that the status information of other processes is not consumed. Historical implementations use the *wait()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001 until *wait()* returns the requested process ID or finds that the requested

40172 process does not exist. Because this means that a shell script could not reliably get the status of  
40173 all background children if a second background job was ever started before the first job finished,  
40174 it is recommended that the *wait* utility use a method such as the functionality provided by the  
40175 *waitpid()* function.

40176 The ability to wait for multiple *pid* operands was adopted from the KornShell.

40177 This new functionality was added because it is needed to determine the exit status of any  
40178 asynchronous list accurately. The only compatibility problem that this change creates is for a  
40179 script like

```
40180 while sleep 60 do
40181 job& echo Job started $(date) as $! done
```

40182 which causes the shell to monitor all of the jobs started until the script terminates or runs out of  
40183 memory. This would not be a problem if the loop did not reference "\$!" or if the script would  
40184 occasionally *wait* for jobs it started.

#### 40185 **FUTURE DIRECTIONS**

40186 None.

#### 40187 **SEE ALSO**

40188 Chapter 2 (on page 29), *kill*, *sh*, the System Interfaces volume of IEEE Std 1003.1-2001, *wait()*,  
40189 *waitpid()*

#### 40190 **CHANGE HISTORY**

40191 First released in Issue 2.



40192 **NAME**

40193           wc — word, line, and byte or character count

40194 **SYNOPSIS**40195           wc [-c|-m] [-lw] [*file*...]40196 **DESCRIPTION**40197           The *wc* utility shall read one or more input files and, by default, write the number of <newline>s,  
40198           words, and bytes contained in each input file to the standard output.40199           The utility also shall write a total count for all named files, if more than one input file is  
40200           specified.40201           The *wc* utility shall consider a *word* to be a non-zero-length string of characters delimited by  
40202           white space.40203 **OPTIONS**40204           The *wc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2,  
40205           Utility Syntax Guidelines.

40206           The following options shall be supported:

40207           **-c**           Write to the standard output the number of bytes in each input file.40208           **-l**           Write to the standard output the number of <newline>s in each input file.40209           **-m**           Write to the standard output the number of characters in each input file.40210           **-w**           Write to the standard output the number of words in each input file.40211           When any option is specified, *wc* shall report only the information requested by the specified  
40212           options.40213 **OPERANDS**

40214           The following operand shall be supported:

40215           *file*           A pathname of an input file. If no *file* operands are specified, the standard input  
40216           shall be used.40217 **STDIN**40218           The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
40219           section.40220 **INPUT FILES**

40221           The input files may be of any type.

40222 **ENVIRONMENT VARIABLES**40223           The following environment variables shall affect the execution of *wc*:40224           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
40225           (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
40226           Internationalization Variables for the precedence of internationalization variables  
40227           used to determine the values of locale categories.)40228           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
40229           internationalization variables.40230           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
40231           characters (for example, single-byte as opposed to multi-byte characters in  
40232           arguments and input files) and which characters are defined as white space  
40233           characters.

40234 **LC\_MESSAGES**

40235 Determine the locale that should be used to affect the format and contents of  
 40236 diagnostic messages written to standard error and informative messages written to  
 40237 standard output.

40238 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

40239 **ASYNCHRONOUS EVENTS**

40240 Default.

40241 **STDOUT**

40242 By default, the standard output shall contain an entry for each input file of the form:

40243 "%d %d %d %s\n", <newlines>, <words>, <bytes>, <file>

40244 If the **-m** option is specified, the number of characters shall replace the <bytes> field in this  
 40245 format.

40246 If any options are specified and the **-l** option is not specified, the number of <newline>s shall  
 40247 not be written.

40248 If any options are specified and the **-w** option is not specified, the number of words shall not be  
 40249 written.

40250 If any options are specified and neither **-c** nor **-m** is specified, the number of bytes or characters  
 40251 shall not be written.

40252 If no input *file* operands are specified, no name shall be written and no <blank>s preceding the  
 40253 pathname shall be written.

40254 If more than one input *file* operand is specified, an additional line shall be written, of the same  
 40255 format as the other lines, except that the word **total** (in the POSIX locale) shall be written instead  
 40256 of a pathname and the total of each column shall be written as appropriate. Such an additional  
 40257 line, if any, is written at the end of the output.

40258 **STDERR**

40259 The standard error shall be used only for diagnostic messages.

40260 **OUTPUT FILES**

40261 None.

40262 **EXTENDED DESCRIPTION**

40263 None.

40264 **EXIT STATUS**

40265 The following exit values shall be returned:

40266 0 Successful completion.

40267 >0 An error occurred.

40268 **CONSEQUENCES OF ERRORS**

40269 Default.

**40270 APPLICATION USAGE**

40271 The **-m** option is not a switch, but an option at the same level as **-c**. Thus, to produce the full  
40272 default output with character counts instead of bytes, the command required is:

40273 `wc -mlw`

**40274 EXAMPLES**

40275 None.

**40276 RATIONALE**

40277 The output file format pseudo-*printf()* string differs from the System V version of *wc*:

40278 `"%7d%7d%7d %s\n"`

40279 which produces possibly ambiguous and unparseable results for very large files, as it assumes no  
40280 number shall exceed six digits.

40281 Some historical implementations use only <space>, <tab>, and <newline> as word separators.  
40282 The equivalent of the ISO C standard *isspace()* function is more appropriate.

40283 The **-c** option stands for “character” count, even though it counts bytes. This stems from the  
40284 sometimes erroneous historical view that bytes and characters are the same size. Due to  
40285 international requirements, the **-m** option (reminiscent of “multi-byte”) was added to obtain  
40286 actual character counts.

40287 Early proposals only specified the results when input files were text files. The current  
40288 specification more closely matches historical practice. (Bytes, words, and <newline>s are  
40289 counted separately and the results are written when an end-of-file is detected.)

40290 Historical implementations of the *wc* utility only accepted one argument to specify the options  
40291 **-c**, **-l**, and **-w**. Some of them also had multiple occurrences of an option cause the  
40292 corresponding count to be written multiple times and had the order of specification of the  
40293 options affect the order of the fields on output, but did not document either of these. Because  
40294 common usage either specifies no options or only one option, and because none of this was  
40295 documented, the changes required by this volume of IEEE Std 1003.1-2001 should not break  
40296 many historical applications (and do not break any historical conforming applications).

**40297 FUTURE DIRECTIONS**

40298 None.

**40299 SEE ALSO**

40300 *cksum*

**40301 CHANGE HISTORY**

40302 First released in Issue 2.

40303 **NAME**40304           what — identify SCCS files (**DEVELOPMENT**)40305 **SYNOPSIS**40306 xSI        what [-s] *file...*

40307

40308 **DESCRIPTION**

40309       The *what* utility shall search the given files for all occurrences of the pattern that *get* (see *get*)  
 40310       substitutes for the %Z% keyword ("@(#) ") and shall write to standard output what follows  
 40311       until the first occurrence of one of the following:

40312           "    &gt;   newline   \    NUL

40313 **OPTIONS**

40314       The *what* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 40315       12.2, Utility Syntax Guidelines.

40316       The following option shall be supported:

40317       -s           Quit after finding the first occurrence of the pattern in each file.

40318 **OPERANDS**

40319       The following operands shall be supported:

40320       *file*           A pathname of a file to search.40321 **STDIN**

40322       Not used.

40323 **INPUT FILES**

40324       The input files shall be of any file type.

40325 **ENVIRONMENT VARIABLES**40326       The following environment variables shall affect the execution of *what*:

40327       *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 40328                        (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 40329       Internationalization Variables for the precedence of internationalization variables  
 40330       used to determine the values of locale categories.)

40331       *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
 40332       internationalization variables.

40333       *LC\_CTYPE*     Determine the locale for the interpretation of sequences of bytes of text data as  
 40334       characters (for example, single-byte as opposed to multi-byte characters in  
 40335       arguments and input files).

40336       *LC\_MESSAGES*

40337                       Determine the locale that should be used to affect the format and contents of  
 40338       diagnostic messages written to standard error.

40339       *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.40340 **ASYNCHRONOUS EVENTS**

40341       Default.

40342 **STDOUT**40343       The standard output shall consist of the following for each *file* operand:

40344       "%s:\n\t%s\n", &lt;pathname&gt;, &lt;identification string&gt;

40345 **STDERR**

40346           The standard error shall be used only for diagnostic messages.

40347 **OUTPUT FILES**

40348           None.

40349 **EXTENDED DESCRIPTION**

40350           None.

40351 **EXIT STATUS**

40352           The following exit values shall be returned:

40353           0    Any matches were found.

40354           1    Otherwise.

40355 **CONSEQUENCES OF ERRORS**

40356           Default.

40357 **APPLICATION USAGE**

40358           The *what* utility is intended to be used in conjunction with the SCCS command *get*, which automatically inserts identifying information, but it can also be used where the information is inserted by any other means.

40361           When the string "@(#)" is included in a library routine in a shared library, it might not be found in an **a.out** file using that library routine.

40363 **EXAMPLES**

40364           If the C-language program in file **f.c** contains:

40365           char ident[] = "@(#)identification information";

40366           and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

40367           what f.c f.o a.out

40368           writes:

40369           f.c:

40370               identification information

40371               ...

40372           f.o:

40373               identification information

40374               ...

40375           a.out:

40376               identification information

40377               ...

40378 **RATIONALE**

40379           None.

40380 **FUTURE DIRECTIONS**

40381           None.

40382 **SEE ALSO**

40383           *get*

40384 **CHANGE HISTORY**

40385           First released in Issue 2.

40386 **NAME**

40387       who — display who is on the system

40388 **SYNOPSIS**

40389 UP       who [-mTu]

40390 XSI       who [-mu]-s [-bHlprt] [*file*]40391       who [-mTu] [-abdHlprt] [*file*]40392       who -q [*file*]

40393       who am i

40394       who am I

40395

40396 **DESCRIPTION**40397       The *who* utility shall list various pieces of information about accessible users. The domain of  
40398 accessibility is implementation-defined.40399 XSI       Based on the options given, *who* can also list the user's name, terminal line, login time, elapsed  
40400 time since activity occurred on the line, and the process ID of the command interpreter for each  
40401 current system user.40402 **OPTIONS**40403       The *who* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
40404 12.2, Utility Syntax Guidelines.40405       The following options shall be supported. The metavariables, such as <*line*>, refer to fields  
40406 described in the STDOUT section.40407 XSI       **-a**       Process the implementation-defined database or named file with the **-b**, **-d**, **-l**, **-p**,  
40408 **-r**, **-t**, **-T** and **-u** options turned on.40409 XSI       **-b**       Write the time and date of the last reboot.40410 XSI       **-d**       Write a list of all processes that have expired and not been respawned by the *init*  
40411 system process. The <*exit*> field shall appear for dead processes and contain the  
40412 termination and exit values of the dead process. This can be useful in determining  
40413 why a process terminated.40414 XSI       **-H**       Write column headings above the regular output.40415 XSI       **-l**       (The letter ell.) List only those lines on which the system is waiting for someone to  
40416 login. The <*name*> field shall be **LOGIN** in such cases. Other fields shall be the  
40417 same as for user entries except that the <*state*> field does not exist.40418       **-m**       Output only information about the current terminal.40419 XSI       **-p**       List any other process that is currently active and has been previously spawned by  
40420 *init*.40421 XSI       **-q**       (Quick.) List only the names and the number of users currently logged on. When  
40422 this option is used, all other options shall be ignored.40423 XSI       **-r**       Write the current *run-level* of the *init* process.40424 XSI       **-s**       List only the <*name*>, <*line*>, and <*time*> fields. This is the default case.40425 XSI       **-t**       Indicate the last change to the system clock.

40426       **-T**       Show the state of each terminal, as described in the STDOUT section.

40427       **-u**       Write “idle time” for each displayed user in addition to any other information. The  
 40428       idle time is the time since any activity occurred on the user’s terminal. The method  
 40429 XSI       of determining this is unspecified. This option shall list only those users who are  
 40430       currently logged in. The *<name>* is the user’s login name. The *<line>* is the name of  
 40431       the line as found in the directory */dev*. The *<time>* is the time that the user logged  
 40432       in. The *<activity>* is the number of hours and minutes since activity last occurred  
 40433       on that particular line. A dot indicates that the terminal has seen activity in the last  
 40434       minute and is therefore “current”. If more than twenty-four hours have elapsed or  
 40435       the line has not been used since boot time, the entry shall be marked *<old>*. This  
 40436       field is useful when trying to determine whether a person is working at the  
 40437       terminal or not. The *<pid>* is the process ID of the user’s login process.

#### 40438 OPERANDS

40439 XSI       The following operands shall be supported:

40440       **am i, am I**   In the POSIX locale, limit the output to describing the invoking user, equivalent to  
 40441       the **-m** option. The **am** and **i** or **I** must be separate arguments.

40442       *file*       Specify a pathname of a file to substitute for the implementation-defined database  
 40443       of logged-on users that *who* uses by default.

#### 40444 STDIN

40445       Not used.

#### 40446 INPUT FILES

40447       None.

#### 40448 ENVIRONMENT VARIABLES

40449       The following environment variables shall affect the execution of *who*:

40450       **LANG**       Provide a default value for the internationalization variables that are unset or null.  
 40451       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 40452       Internationalization Variables for the precedence of internationalization variables  
 40453       used to determine the values of locale categories.)

40454       **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 40455       internationalization variables.

40456       **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
 40457       characters (for example, single-byte as opposed to multi-byte characters in  
 40458       arguments).

40459       **LC\_MESSAGES**  
 40460       Determine the locale that should be used to affect the format and contents of  
 40461       diagnostic messages written to standard error.

40462       **LC\_TIME**     Determine the locale used for the format and contents of the date and time strings.

40463 XSI       **NLSPATH**   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40464       **TZ**         Determine the timezone used when writing date and time information. If *TZ* is  
 40465       unset or null, an unspecified default timezone shall be used.

#### 40466 ASYNCHRONOUS EVENTS

40467       Default.

**40468 STDOUT**

40469 The *who* utility shall write its default format to the standard output in an implementation-  
 40470 defined format, subject only to the requirement of containing the information described above.

40471 XSI OF XSI-conformant systems shall write the default information to the standard output in the  
 40472 following general format:

40473 `<name>[<state>]<line><time>[<activity>] [<pid>] [<comment>] [<exit>]`

40474 The following format shall be used for the **-T** option:

40475 `"%s %c %s %s\n" <name>, <terminal state>, <terminal name>,  
 40476 <time of login>`

40477 where *<terminal state>* is one of the following characters:

- 40478 + The terminal allows write access to other users.
- 40479 - The terminal denies write access to other users.
- 40480 ? The terminal write-access state cannot be determined.

40481 In the POSIX locale, the *<time of login>* shall be equivalent in format to the output of:

40482 `date +"%b %e %H:%M"`

40483 If the **-u** option is used with **-T**, the idle time shall be added to the end of the previous format in  
 40484 an unspecified format.

**40485 STDERR**

40486 The standard error shall be used only for diagnostic messages.

**40487 OUTPUT FILES**

40488 None.

**40489 EXTENDED DESCRIPTION**

40490 None.

**40491 EXIT STATUS**

40492 The following exit values shall be returned:

- 40493 0 Successful completion.
- 40494 >0 An error occurred.

**40495 CONSEQUENCES OF ERRORS**

40496 Default.

**40497 APPLICATION USAGE**

40498 The name *init* used for the system process is the most commonly used on historical systems, but  
 40499 it may vary.

40500 The “domain of accessibility” referred to is a broad concept that permits interpretation either on  
 40501 a very secure basis or even to allow a network-wide implementation like the historical *rwho*.

**40502 EXAMPLES**

40503 None.

**40504 RATIONALE**

40505 Due to differences between historical implementations, the base options provided were a  
 40506 compromise to allow users to work with those functions. The standard developers also  
 40507 considered removing all the options, but felt that these options offered users valuable  
 40508 functionality. Additional options to match historical systems are available on XSI-conformant



- 40509 systems.
- 40510 It is recognized that the *who* command may be of limited usefulness, especially in a multi-level  
40511 secure environment. The standard developers considered, however, that having some standard  
40512 method of determining the “accessibility” of other users would aid user portability.
- 40513 No format was specified for the default *who* output for systems not supporting the XSI  
40514 Extension. In such a user-oriented command, designed only for human use, this was not  
40515 considered to be a deficiency.
- 40516 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, and *write* require  
40517 that they use the same format.
- 40518 It is acceptable for an implementation to produce no output for an invocation of *who* **mil**.
- 40519 **FUTURE DIRECTIONS**
- 40520 None.
- 40521 **SEE ALSO**
- 40522 *mesg*
- 40523 **CHANGE HISTORY**
- 40524 First released in Issue 2.
- 40525 **Issue 6**
- 40526 This utility is marked as part of the User Portability Utilities option.
- 40527 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

40528 **NAME**

40529 write — write to another user

40530 **SYNOPSIS**40531 UP write *user\_name* [*terminal*]

40532

40533 **DESCRIPTION**40534 The *write* utility shall read lines from the user's standard input and write them to the terminal of  
40535 another user. When first invoked, it shall write the message:40536 **Message from** *sender-login-id* (*sending-terminal*) [*date*] ...40537 to *user\_name*. When it has successfully completed the connection, the sender's terminal shall be  
40538 alerted twice to indicate that what the sender is typing is being written to the recipient's  
40539 terminal.

40540 If the recipient wants to reply, this can be accomplished by typing:

40541 write *sender-login-id* [*sending-terminal*]40542 upon receipt of the initial message. Whenever a line of input as delimited by an NL, EOF, or EOL  
40543 special character (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General  
40544 Terminal Interface) is accumulated while in canonical input mode, the accumulated data shall be  
40545 written on the other user's terminal. Characters shall be processed as follows:

- 40546 • Typing <alert> shall write the alert character to the recipient's terminal.
- 40547 • Typing the erase and kill characters shall affect the sender's terminal in the manner described  
40548 by the **termios** interface in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11,  
40549 General Terminal Interface.
- 40550 • Typing the interrupt or end-of-file characters shall cause *write* to write an appropriate  
40551 message ("EOT\n" in the POSIX locale) to the recipient's terminal and exit.
- 40552 • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those characters  
40553 to be sent to the recipient's terminal.
- 40554 • When and only when the *stty* **ixten** local mode is enabled, the existence and processing of  
40555 additional special control characters and multi-byte or single-byte functions is  
40556 implementation-defined.
- 40557 • Typing other non-printable characters shall cause implementation-defined sequences of  
40558 printable characters to be written to the recipient's terminal.

40559 To write to a user who is logged in more than once, the *terminal* argument can be used to indicate  
40560 which terminal to write to; otherwise, the recipient's terminal is selected in an implementation-  
40561 defined manner and an informational message is written to the sender's standard output,  
40562 indicating which terminal was chosen.40563 Permission to be a recipient of a *write* message can be denied or granted by use of the *mesg*  
40564 utility. However, a user's privilege may further constrain the domain of accessibility of other  
40565 users' terminals. The *write* utility shall fail when the user lacks the appropriate privileges to  
40566 perform the requested action.40567 **OPTIONS**

40568 None.

40569 **OPERANDS**

40570 The following operands shall be supported:

40571 *user\_name* Login name of the person to whom the message shall be written. The application  
 40572 shall ensure that this operand is of the form returned by the *who* utility.

40573 *terminal* Terminal identification in the same format provided by the *who* utility.

40574 **STDIN**

40575 Lines to be copied to the recipient's terminal are read from standard input.

40576 **INPUT FILES**

40577 None.

40578 **ENVIRONMENT VARIABLES**40579 The following environment variables shall affect the execution of *write*:

40580 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 40581 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 40582 Internationalization Variables for the precedence of internationalization variables  
 40583 used to determine the values of locale categories.)

40584 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 40585 internationalization variables.

40586 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 40587 characters (for example, single-byte as opposed to multi-byte characters in  
 40588 arguments and input files). If the recipient's locale does not use an *LC\_CTYPE*  
 40589 equivalent to the sender's, the results are undefined.

40590 *LC\_MESSAGES*

40591 Determine the locale that should be used to affect the format and contents of  
 40592 diagnostic messages written to standard error and informative messages written to  
 40593 standard output.

40594 *XS1* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40595 **ASYNCHRONOUS EVENTS**

40596 If an interrupt signal is received, *write* shall write an appropriate message on the recipient's  
 40597 terminal and exit with a status of zero. It shall take the standard action for all other signals.

40598 **STDOUT**

40599 An informational message shall be written to standard output if a recipient is logged in more  
 40600 than once.

40601 **STDERR**

40602 The standard error shall be used only for diagnostic messages.

40603 **OUTPUT FILES**

40604 The recipient's terminal is used for output.

40605 **EXTENDED DESCRIPTION**

40606 None.

40607 **EXIT STATUS**

40608 The following exit values shall be returned:

40609 0 Successful completion.

40610 &gt;0 The addressed user is not logged on or the addressed user denies permission.

**40611 CONSEQUENCES OF ERRORS**

40612 Default.

**40613 APPLICATION USAGE**

40614 The *talk* utility is considered by some users to be a more usable utility on full-screen terminals.

**40615 EXAMPLES**

40616 None.

**40617 RATIONALE**

40618 The *write* utility was included in this volume of IEEE Std 1003.1-2001 since it can be  
40619 implemented on all terminal types. The standard developers considered the *talk* utility, which  
40620 cannot be implemented on certain terminals, to be a “better” communications interface. Both of  
40621 these programs are in widespread use on historical implementations. Therefore, the standard  
40622 developers decided that both utilities should be specified.

40623 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
40624 require that they all use or accept the same format.

**40625 FUTURE DIRECTIONS**

40626 None.

**40627 SEE ALSO**

40628 *mesg*, *talk*, *who*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General  
40629 Terminal Interface

**40630 CHANGE HISTORY**

40631 First released in Issue 2.

**40632 Issue 5**

40633 The FUTURE DIRECTIONS section is added.

**40634 Issue 6**

40635 This utility is marked as part of the User Portability Utilities option.

40636 The normative text is reworded to avoid use of the term “must” for application requirements.

40637 **NAME**

40638 xargs — construct argument lists and invoke utility

40639 **SYNOPSIS**

40640 xsi xargs [-t] [-p]] [-E *eofstr*] [-I *replstr*] [-L *number*] [-n *number* [-x]]  
 40641 [-s *size*] [*utility* [*argument...*]]

40642 **DESCRIPTION**

40643 The *xargs* utility shall construct a command line consisting of the *utility* and *argument* operands  
 40644 specified followed by as many arguments read in sequence from standard input as fit in length  
 40645 and number constraints specified by the options. The *xargs* utility shall then invoke the  
 40646 constructed command line and wait for its completion. This sequence shall be repeated until one  
 40647 of the following occurs:

- 40648 • An end-of-file condition is detected on standard input.
- 40649 • The logical end-of-file string (see the **-E** *eofstr* option) is found on standard input after  
 40650 double-quote processing, apostrophe processing, and backslash escape processing (see next  
 40651 paragraph).
- 40652 • An invocation of a constructed command line returns an exit status of 255.

40653 The application shall ensure that arguments in the standard input are separated by unquoted  
 40654 <blank>s, unescaped <blank>s, or <newline>s. A string of zero or more non-double-quote ( ' " ' )  
 40655 characters and non-<newline>s can be quoted by enclosing them in double-quotes. A string of  
 40656 zero or more non-apostrophe ( ' ' ' ) characters and non-<newline>s can be quoted by enclosing  
 40657 them in apostrophes. Any unquoted character can be escaped by preceding it with a backslash.  
 40658 The utility named by *utility* shall be executed one or more times until the end-of-file is reached  
 40659 or the logical end-of file string is found. The results are unspecified if the utility named by *utility*  
 40660 attempts to read from its standard input.

40661 The generated command line length shall be the sum of the size in bytes of the utility name and  
 40662 each argument treated as strings, including a null byte terminator for each of these strings. The  
 40663 *xargs* utility shall limit the command line length such that when the command line is invoked,  
 40664 the combined argument and environment lists (see the *exec* family of functions in the System  
 40665 Interfaces volume of IEEE Std 1003.1-2001) shall not exceed {ARG\_MAX}-2048 bytes. Within  
 40666 this constraint, if neither the **-n** nor the **-s** option is specified, the default command line length  
 40667 shall be at least {LINE\_MAX}.

40668 **OPTIONS**

40669 The *xargs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 40670 12.2, Utility Syntax Guidelines.

40671 The following options shall be supported:

40672 **-E** *eofstr* Use *eofstr* as the logical end-of-file string. If **-E** is not specified, it is unspecified  
 40673 whether the logical end-of-file string is the underscore character ( ' \_ ' ) or the end-  
 40674 of-file string capability is disabled. When *eofstr* is the null string, the logical end-  
 40675 of-file string capability shall be disabled and underscore characters shall be taken  
 40676 literally.

40677 xsi **-I** *replstr* Insert mode: *utility* is executed for each line from standard input, taking the entire  
 40678 line as a single argument, inserting it in *arguments* for each occurrence of *replstr*. A  
 40679 maximum of five arguments in *arguments* can each contain one or more instances  
 40680 of *replstr*. Any <blank>s at the beginning of each line shall be ignored.  
 40681 Constructed arguments cannot grow larger than 255 bytes. Option **-x** shall be  
 40682 forced on.

|           |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 40683 XSI | <b>-L number</b> | The <i>utility</i> shall be executed for each non-empty <i>number</i> lines of arguments from standard input. The last invocation of <i>utility</i> shall be with fewer lines of arguments if fewer than <i>number</i> remain. A line is considered to end with the first <newline> unless the last character of the line is a <blank>; a trailing <blank> signals continuation to the next non-empty line, inclusive. The <b>-L</b> and <b>-n</b> options are mutually-exclusive; the last one specified shall take effect. |
| 40684     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40685     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40686     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40687     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40688     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40689     | <b>-n number</b> | Invoke <i>utility</i> using as many standard input arguments as possible, up to <i>number</i> (a positive decimal integer) arguments maximum. Fewer arguments shall be used if:                                                                                                                                                                                                                                                                                                                                              |
| 40690     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40691     |                  | <ul style="list-style-type: none"> <li>• The command line length accumulated exceeds the size specified by the <b>-s</b> option (or {LINE_MAX} if there is no <b>-s</b> option).</li> </ul>                                                                                                                                                                                                                                                                                                                                  |
| 40692     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40693     |                  | <ul style="list-style-type: none"> <li>• The last iteration has fewer than <i>number</i>, but not zero, operands remaining.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                       |
| 40694     | <b>-p</b>        | Prompt mode: the user is asked whether to execute <i>utility</i> at each invocation. Trace mode ( <b>-t</b> ) is turned on to write the command instance to be executed, followed by a prompt to standard error. An affirmative response read from <b>/dev/tty</b> shall execute the command; otherwise, that particular invocation of <i>utility</i> shall be skipped.                                                                                                                                                      |
| 40695     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40696     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40697     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40698     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40699     | <b>-s size</b>   | Invoke <i>utility</i> using as many standard input arguments as possible yielding a command line length less than <i>size</i> (a positive decimal integer) bytes. Fewer arguments shall be used if:                                                                                                                                                                                                                                                                                                                          |
| 40700     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40701     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40702     |                  | <ul style="list-style-type: none"> <li>• The total number of arguments exceeds that specified by the <b>-n</b> option.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                            |
| 40703 XSI |                  | <ul style="list-style-type: none"> <li>• The total number of lines exceeds that specified by the <b>-L</b> option.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                |
| 40704     |                  | <ul style="list-style-type: none"> <li>• End-of-file is encountered on standard input before <i>size</i> bytes are accumulated.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                   |
| 40705     |                  | Values of <i>size</i> up to at least {LINE_MAX} bytes shall be supported, provided that the constraints specified in the DESCRIPTION are met. It shall not be considered an error if a value larger than that supported by the implementation or exceeding the constraints specified in the DESCRIPTION is given; <i>xargs</i> shall use the largest value it supports within the constraints.                                                                                                                               |
| 40706     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40707     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40708     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40709     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40710     | <b>-t</b>        | Enable trace mode. Each generated command line shall be written to standard error just prior to invocation.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 40711     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40712     | <b>-x</b>        | Terminate if a command line containing <i>number</i> arguments (see the <b>-n</b> option above) or <i>number</i> lines (see the <b>-L</b> option above) will not fit in the implied or specified size (see the <b>-s</b> option above).                                                                                                                                                                                                                                                                                      |
| 40713 XSI |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 40714     |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## 40715 OPERANDS

40716 The following operands shall be supported:

|       |                 |                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 40717 | <i>utility</i>  | The name of the utility to be invoked, found by search path using the <i>PATH</i> environment variable, described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables. If <i>utility</i> is omitted, the default shall be the <i>echo</i> utility. If the <i>utility</i> operand names any of the special built-in utilities in Section 2.14 (on page 64), the results are undefined. |
| 40718 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 40719 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 40720 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 40721 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 40722 | <i>argument</i> | An initial option or operand for the invocation of <i>utility</i> .                                                                                                                                                                                                                                                                                                                                                        |

## 40723 STDIN

40724 The standard input shall be a text file. The results are unspecified if an end-of-file condition is detected immediately following an escaped <newline>.

40725

40726 **INPUT FILES**

40727 The file **/dev/tty** shall be used to read responses required by the **-p** option.

40728 **ENVIRONMENT VARIABLES**

40729 The following environment variables shall affect the execution of *xargs*:

40730 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 40731 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 40732 Internationalization Variables for the precedence of internationalization variables  
 40733 used to determine the values of locale categories.)

40734 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 40735 internationalization variables.

40736 **LC\_COLLATE**  
 40737 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 40738 character collating elements used in the extended regular expression defined for  
 40739 the **yesexpr** locale keyword in the **LC\_MESSAGES** category.

40740 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 40741 characters (for example, single-byte as opposed to multi-byte characters in  
 40742 arguments and input files) and the behavior of character classes used in the  
 40743 extended regular expression defined for the **yesexpr** locale keyword in the  
 40744 **LC\_MESSAGES** category.

40745 **LC\_MESSAGES**  
 40746 Determine the locale for the processing of affirmative responses and that should be  
 40747 used to affect the format and contents of diagnostic messages written to standard  
 40748 error.

40749 **XSI** **NLS\_PATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

40750 **PATH** Determine the location of *utility*, as described in the Base Definitions volume of  
 40751 IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

40752 **ASYNCHRONOUS EVENTS**

40753 Default.

40754 **STDOUT**

40755 Not used.

40756 **STDERR**

40757 The standard error shall be used for diagnostic messages and the **-t** and **-p** options. If the **-t**  
 40758 option is specified, the *utility* and its constructed argument list shall be written to standard error,  
 40759 as it will be invoked, prior to invocation. If **-p** is specified, a prompt of the following format  
 40760 shall be written (in the POSIX locale):

40761 " ? . . . "

40762 at the end of the line of the output from **-t**.

40763 **OUTPUT FILES**

40764 None.

40765 **EXTENDED DESCRIPTION**

40766 None.

40767 **EXIT STATUS**

40768 The following exit values shall be returned:

- 40769           0     All invocations of *utility* returned exit status zero.
- 40770           1-125   A command line meeting the specified requirements could not be assembled, one or  
40771 more of the invocations of *utility* returned a non-zero exit status, or some other error  
40772 occurred.
- 40773           126    The utility specified by *utility* was found but could not be invoked.
- 40774           127    The utility specified by *utility* could not be found.

40775 **CONSEQUENCES OF ERRORS**

40776 If a command line meeting the specified requirements cannot be assembled, the utility cannot be  
40777 invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits  
40778 with exit status 255, the *xargs* utility shall write a diagnostic message and exit without  
40779 processing any remaining input.

40780 **APPLICATION USAGE**

40781 The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no  
40782 further invocations using the current data stream will succeed. Thus, *utility* should explicitly *exit*  
40783 with an appropriate value to avoid accidentally returning with 255.

40784 Note that input is parsed as lines; <blank>s separate arguments. If *xargs* is used to bundle output  
40785 of commands like *find dir -print* or *ls* into commands to be executed, unexpected results are  
40786 likely if any filenames contain any <blank>s or <newline>s. This can be fixed by using *find* to  
40787 call a script that converts each file found into a quoted string that is then piped to *xargs*. Note  
40788 that the quoting rules used by *xargs* are not the same as in the shell. They were not made  
40789 consistent here because existing applications depend on the current rules and the shell syntax is  
40790 not fully compatible with it. An easy rule that can be used to transform any string into a quoted  
40791 form that *xargs* interprets correctly is to precede each character in the string with a backslash.

40792 On implementations with a large value for {ARG\_MAX}, *xargs* may produce command lines  
40793 longer than {LINE\_MAX}. For invocation of utilities, this is not a problem. If *xargs* is being used  
40794 to create a text file, users should explicitly set the maximum command line length with the *-s*  
40795 option.

40796 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
40797 an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
40798 utility exited with an error indication”. The value 127 was chosen because it is not commonly  
40799 used for other meanings; most utilities use small values for “normal error conditions” and the  
40800 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
40801 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
40802 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
40803 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
40804 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
40805 any other reason.

40806 **EXAMPLES**

- 40807           1. The following command combines the output of the parenthesised commands onto one  
40808 line, which is then written to the end-of-file **log**:
- 40809           (logname; date; printf "%s\n" "\$0 \$\*") | xargs >>log
- 40810           2. The following command invokes *diff* with successive pairs of arguments originally typed  
40811 as command line arguments (assuming there are no embedded <blank>s in the elements of  
40812 the original argument list):



```

40813 printf "%s\n" "$*" | xargs -n 2 -x diff
40814
40815 3. In the following commands, the user is asked which files in the current directory are to be
 archived. The files are archived into arch; a, one at a time, or b, many at a time.
40816
40817 a. ls | xargs -p -L 1 ar -r arch
40818
40819 b. ls | xargs -p -L 1 | xargs ar -r arch
40820
40821 4. The following executes with successive pairs of arguments originally typed as command
40822 line arguments:
40823
40824 echo $* | xargs -n 2 diff
40825
40826 5. On XSI-conformant systems, the following moves all files from directory $1 to directory $2,
40827 and echoes each move command just before doing it:
40828
40829 ls $1 | xargs -I {} -t mv $1/{} $2/{}

```

#### 40824 RATIONALE

40825 The *xargs* utility was usually found only in System V-based systems; BSD systems included an  
 40826 *apply* utility that provided functionality similar to *xargs -n number*. The SVID lists *xargs* as a  
 40827 software development extension. This volume of IEEE Std 1003.1-2001 does not share the view  
 40828 that it is used only for development, and therefore it is not optional.

40829 The classic application of the *xargs* utility is in conjunction with the *find* utility to reduce the  
 40830 number of processes launched by a simplistic use of the *find -exec* combination. The *xargs* utility  
 40831 is also used to enforce an upper limit on memory required to launch a process. With this basis in  
 40832 mind, this volume of IEEE Std 1003.1-2001 selected only the minimal features required.

40833 Although the 255 exit status is mostly an accident of historical implementations, it allows a  
 40834 utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the  
 40835 current data stream shall succeed. Any non-zero exit status from a utility falls into the 1-125  
 40836 range when *xargs* exits. There is no statement of how the various non-zero utility exit status  
 40837 codes are accumulated by *xargs*. The value could be the addition of all codes, their highest  
 40838 value, the last one received, or a single value such as 1. Since no algorithm is arguably better  
 40839 than the others, and since many of the standard utilities say little more (portably) than  
 40840 “pass/fail”, no new algorithm was invented.

40841 Several other *xargs* options were withdrawn because simple alternatives already exist within this  
 40842 volume of IEEE Std 1003.1-2001. For example, the *-i replstr* option can be just as efficiently  
 40843 performed using a shell **for** loop. Since *xargs* calls an *exec* function with each input line, the *-i*  
 40844 option does not usually exploit the grouping capabilities of *xargs*.

40845 The requirement that *xargs* never produces command lines such that invocation of *utility* is  
 40846 within 2048 bytes of hitting the POSIX *exec* {ARG\_MAX} limitations is intended to guarantee  
 40847 that the invoked utility has room to modify its environment variables and command line  
 40848 arguments and still be able to invoke another utility. Note that the minimum {ARG\_MAX}  
 40849 allowed by the System Interfaces volume of IEEE Std 1003.1-2001 is 4096 bytes and the  
 40850 minimum value allowed by this volume of IEEE Std 1003.1-2001 is 2048 bytes; therefore, the  
 40851 2048 bytes difference seems reasonable. Note, however, that *xargs* may never be able to invoke a  
 40852 utility if the environment passed in to *xargs* comes close to using {ARG\_MAX} bytes.

40853 The version of *xargs* required by this volume of IEEE Std 1003.1-2001 is required to wait for the  
 40854 completion of the invoked command before invoking another command. This was done because  
 40855 historical scripts using *xargs* assumed sequential execution. Implementations wanting to provide  
 40856 parallel operation of the invoked utilities are encouraged to add an option enabling parallel  
 40857 invocation, but should still wait for termination of all of the children before *xargs* terminates  
 40858 normally.

The **-e** option was omitted from the ISO POSIX-2:1993 standard in the belief that the *eofstr* option-argument was recognized only when it was on a line by itself and before quote and escape processing were performed, and that the logical end-of-file processing was only enabled if a **-e** option was specified. In that case, a simple *sed* script could be used to duplicate the **-e** functionality. Further investigation revealed that:

- The logical end-of-file string was checked for after quote and escape processing, making a *sed* script that provided equivalent functionality much more difficult to write.
- The default was to perform logical end-of-file processing with an underscore as the logical end-of-file string.

To correct this misunderstanding, the **-E eofstr** option was adopted from the X/Open Portability Guide. Users should note that the description of the **-E** option matches historical documentation of the **-e** option (which was not adopted because it did not support the Utility Syntax Guidelines), by saying that if *eofstr* is the null string, logical end-of-file processing is disabled. Historical implementations of *xargs* actually did not disable logical end-of-file processing; they treated a null argument found in the input as a logical end-of-file string. (A null *string* argument could be generated using single or double quotes ( ' ' or " "). Since this behavior was not documented historically, it is considered to be a bug.

#### **FUTURE DIRECTIONS**

None.

#### **SEE ALSO**

Chapter 2 (on page 29), *echo*, *find*, the System Interfaces volume of IEEE Std 1003.1-2001, *exec*

#### **CHANGE HISTORY**

First released in Issue 2.

#### **Issue 5**

A second FUTURE DIRECTION is added.

#### **Issue 6**

The obsolescent **-e**, **-i**, and **-l** options are removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **-p** option is added.
- In the INPUT FILES section, the file **/dev/tty** is used to read responses required by the **-p** option.
- The STDERR section is updated to describe the **-p** option.

The description of the **-E** option is aligned with the ISO POSIX-2: 1993 standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

40894 **NAME**40895 yacc — yet another compiler compiler (**DEVELOPMENT**)40896 **SYNOPSIS**40897 CD yacc [-dltv] [-b *file\_prefix*] [-p *sym\_prefix*] *grammar*

40898

40899 **DESCRIPTION**

40900 The *yacc* utility shall read a description of a context-free grammar in *grammar* and write C source  
 40901 code, conforming to the ISO C standard, to a code file, and optionally header information into a  
 40902 header file, in the current directory. The C code shall define a function and related routines and  
 40903 macros for an automaton that executes a parsing algorithm meeting the requirements in  
 40904 **Algorithms** (on page 1074).

40905 The form and meaning of the grammar are described in the EXTENDED DESCRIPTION section.

40906 The C source code and header file shall be produced in a form suitable as input for the C  
 40907 compiler (see *c99*).

40908 **OPTIONS**

40909 The *yacc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-2001, Section  
 40910 12.2, Utility Syntax Guidelines.

40911 The following options shall be supported:

40912 **-b** *file\_prefix* Use *file\_prefix* instead of **y** as the prefix for all output filenames. The code file  
 40913 **y.tab.c**, the header file **y.tab.h** (created when **-d** is specified), and the description  
 40914 file **y.output** (created when **-v** is specified), shall be changed to *file\_prefix.tab.c*,  
 40915 *file\_prefix.tab.h*, and *file\_prefix.output*, respectively.

40916 **-d** Write the header file; by default only the code file is written. The **#define**  
 40917 statements associate the token codes assigned by *yacc* with the user-declared token  
 40918 names. This allows source files other than **y.tab.c** to access the token codes.

40919 **-l** Produce a code file that does not contain any **#line** constructs. If this option is not  
 40920 present, it is unspecified whether the code file or header file contains **#line**  
 40921 directives. This should only be used after the grammar and the associated actions  
 40922 are fully debugged.

40923 **-p** *sym\_prefix* Use *sym\_prefix* instead of **yy** as the prefix for all external names produced by *yacc*.  
 40924 The names affected shall include the functions *yyparse()*, *yylex()*, and *yyerror()*,  
 40925 and the variables *yyval*, *yychar*, and *yydebug*. (In the remainder of this section, the  
 40926 six symbols cited are referenced using their default names only as a notational  
 40927 convenience.) Local names may also be affected by the **-p** option; however, the **-p**  
 40928 option shall not affect **#define** symbols generated by *yacc*.  
 40929

40930 **-t** Modify conditional compilation directives to permit compilation of debugging  
 40931 code in the code file. Runtime debugging statements shall always be contained in  
 40932 the code file, but by default conditional compilation directives prevent their  
 40933 compilation.

40934 **-v** Write a file containing a description of the parser and a report of conflicts  
 40935 generated by ambiguities in the grammar.

40936 **OPERANDS**

40937 The following operand is required:

40938 *grammar* A pathname of a file containing instructions, hereafter called *grammar*, for which a  
 40939 parser is to be created. The format for the grammar is described in the EXTENDED  
 40940 DESCRIPTION section.

40941 **STDIN**

40942 Not used.

40943 **INPUT FILES**

40944 The file *grammar* shall be a text file formatted as specified in the EXTENDED DESCRIPTION  
 40945 section.

40946 **ENVIRONMENT VARIABLES**40947 The following environment variables shall affect the execution of *yacc*:

40948 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 40949 (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 40950 Internationalization Variables for the precedence of internationalization variables  
 40951 used to determine the values of locale categories.)

40952 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 40953 internationalization variables.

40954 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 40955 characters (for example, single-byte as opposed to multi-byte characters in  
 40956 arguments and input files).

40957 *LC\_MESSAGES*

40958 Determine the locale that should be used to affect the format and contents of  
 40959 diagnostic messages written to standard error.

40960 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40961 The *LANG* and *LC\_\** variables affect the execution of the *yacc* utility as stated. The *main()*  
 40962 function defined in **Yacc Library** (on page 1074) shall call:

40963 `setlocale(LC_ALL, "")`

40964 and thus the program generated by *yacc* shall also be affected by the contents of these variables  
 40965 at runtime.

40966 **ASYNCHRONOUS EVENTS**

40967 Default.

40968 **STDOUT**

40969 Not used.

40970 **STDERR**

40971 If shift/reduce or reduce/reduce conflicts are detected in *grammar*, *yacc* shall write a report of  
 40972 those conflicts to the standard error in an unspecified format.

40973 Standard error shall also be used for diagnostic messages.

40974 **OUTPUT FILES**

40975 The code file, the header file, and the description file shall be text files. All are described in the  
 40976 following sections.

**40977 Code File**

40978 This file shall contain the C source code for the `yyparse()` function. It shall contain code for the  
40979 various semantic actions with macro substitution performed on them as described in the  
40980 EXTENDED DESCRIPTION section. It also shall contain a copy of the **#define** statements in the  
40981 header file. If a **%union** declaration is used, the declaration for YYSTYPE shall also be included  
40982 in this file.

**40983 Header File**

40984 The header file shall contain **#define** statements that associate the token numbers with the token  
40985 names. This allows source files other than the code file to access the token codes. If a **%union**  
40986 declaration is used, the declaration for YYSTYPE and an *extern YYSTYPE yylval* declaration shall  
40987 also be included in this file.

**40988 Description File**

40989 The description file shall be a text file containing a description of the state machine  
40990 corresponding to the parser, using an unspecified format. Limits for internal tables (see **Limits**  
40991 (on page 1075)) shall also be reported, in an implementation-defined manner. (Some  
40992 implementations may use dynamic allocation techniques and have no specific limit values to  
40993 report.)

**40994 EXTENDED DESCRIPTION**

40995 The `yacc` command accepts a language that is used to define a grammar for a target language to  
40996 be parsed by the tables and code generated by `yacc`. The language accepted by `yacc` as a  
40997 grammar for the target language is described below using the `yacc` input language itself.

40998 The input *grammar* includes rules describing the input structure of the target language and code  
40999 to be invoked when these rules are recognized to provide the associated semantic action. The  
41000 code to be executed shall appear as bodies of text that are intended to be C-language code. The  
41001 C-language inclusions are presumed to form a correct function when processed by `yacc` into its  
41002 output files. The code included in this way shall be executed during the recognition of the target  
41003 language.

41004 Given a grammar, the `yacc` utility generates the files described in the OUTPUT FILES section.  
41005 The code file can be compiled and linked using `c99`. If the declaration and programs sections of  
41006 the grammar file did not include definitions of `main()`, `yylex()`, and `yyerror()`, the compiled  
41007 output requires linking with externally supplied versions of those functions. Default versions of  
41008 `main()` and `yyerror()` are supplied in the `yacc` library and can be linked in by using the **-ly**  
41009 operand to `c99`. The `yacc` library interfaces need not support interfaces with other than the  
41010 default **yy** symbol prefix. The application provides the lexical analyzer function, `yylex()`; the *lex*  
41011 utility is specifically designed to generate such a routine.

**41012 Input Language**

41013 The application shall ensure that every specification file consists of three sections in order:  
41014 *declarations*, *grammar rules*, and *programs*, separated by double percent signs ("%"). The  
41015 declarations and programs sections can be empty. If the latter is empty, the preceding "%"   
41016 mark separating it from the rules section can be omitted.

41017 The input is free form text following the structure of the grammar defined below.

## Lexical Structure of the Grammar

The <blank>s, <newline>s, and <form-feed>s shall be ignored, except that the application shall ensure that they do not appear in names or multi-character reserved symbols. Comments shall be enclosed in `"/ * . . . */`, and can appear wherever a name is valid.

Names are of arbitrary length, made up of letters, periods (`'.'`), underscores (`'_'`), and non-initial digits. Uppercase and lowercase letters are distinct. Conforming applications shall not use names beginning in **yy** or **YY** since the *yacc* parser uses such names. Many of the names appear in the final output of *yacc*, and thus they should be chosen to conform with any additional rules created by the C compiler to be used. In particular they appear in **#define** statements.

A literal shall consist of a single character enclosed in single-quotes (`' '`). All of the escape sequences supported for character constants by the ISO C standard shall be supported by *yacc*.

The relationship with the lexical analyzer is discussed in detail below.

The application shall ensure that the NUL character is not used in grammar rules or literals.

## Declarations Section

The declarations section is used to define the symbols used to define the target language and their relationship with each other. In particular, much of the additional information required to resolve ambiguities in the context-free grammar for the target language is provided here.

Usually *yacc* assigns the relationship between the symbolic names it generates and their underlying numeric value. The declarations section makes it possible to control the assignment of these values.

It is also possible to keep semantic information associated with the tokens currently on the parse stack in a user-defined C-language **union**, if the members of the union are associated with the various names in the grammar. The declarations section provides for this as well.

The first group of declarators below all take a list of names as arguments. That list can optionally be preceded by the name of a C union member (called a *tag* below) appearing within `'<'` and `'>'`. (As an exception to the typographical conventions of the rest of this volume of IEEE Std 1003.1-2001, in this case `<tag>` does not represent a metavariable, but the literal angle bracket characters surrounding a symbol.) The use of *tag* specifies that the tokens named on this line shall be of the same C type as the union member referenced by *tag*. This is discussed in more detail below.

For lists used to define tokens, the first appearance of a given token can be followed by a positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it for lexical purposes shall be taken to be that number.

The following declares *name* to be a token:

```
%token [<tag>] name [number] [name [number]] . . .
```

If *tag* is present, the C type for all tokens on this line shall be declared to be the type referenced by *tag*. If a positive integer, *number*, follows a *name*, that value shall be assigned to the token.

The following declares *name* to be a token, and assigns precedence to it:

```
%left [<tag>] name [number] [name [number]] . . .
```

```
%right [<tag>] name [number] [name [number]] . . .
```

One or more lines, each beginning with one of these symbols, can appear in this section. All tokens on the same line have the same precedence level and associativity; the lines are in order

41061 of increasing precedence or binding strength. **%left** denotes that the operators on that line are  
 41062 left associative, and **%right** similarly denotes right associative operators. If *tag* is present, it shall  
 41063 declare a C type for *names* as described for **%token**.

41064 The following declares *name* to be a token, and indicates that this cannot be used associatively:

41065 `%nonassoc [<tag>] name [number] [name [number]] ...`

41066 If the parser encounters associative use of this token it reports an error. If *tag* is present, it shall  
 41067 declare a C type for *names* as described for **%token**.

41068 The following declares that union member *names* are non-terminals, and thus it is required to  
 41069 have a *tag* field at its beginning:

41070 `%type <tag> name...`

41071 Because it deals with non-terminals only, assigning a token number or using a literal is also  
 41072 prohibited. If this construct is present, *yacc* shall perform type checking; if this construct is not  
 41073 present, the parse stack shall hold only the **int** type.

41074 Every name used in *grammar* not defined by a **%token**, **%left**, **%right**, or **%nonassoc** declaration  
 41075 is assumed to represent a non-terminal symbol. The *yacc* utility shall report an error for any  
 41076 non-terminal symbol that does not appear on the left side of at least one grammar rule.

41077 Once the type, precedence, or token number of a name is specified, it shall not be changed. If the  
 41078 first declaration of a token does not assign a token number, *yacc* shall assign a token number.  
 41079 Once this assignment is made, the token number shall not be changed by explicit assignment.

41080 The following declarators do not follow the previous pattern.

41081 The following declares the non-terminal *name* to be the *start symbol*, which represents the largest,  
 41082 most general structure described by the grammar rules:

41083 `%start name`

41084 By default, it is the left-hand side of the first grammar rule; this default can be overridden with  
 41085 this declaration.

41086 The following declares the *yacc* value stack to be a union of the various types of values desired:

41087 `%union { body of union (in C) }`

41088 By default, the values returned by actions (see below) and the lexical analyzer shall be of type  
 41089 **int**. The *yacc* utility keeps track of types, and it shall insert corresponding union member names  
 41090 in order to perform strict type checking of the resulting parser.

41091 Alternatively, given that at least one *<tag>* construct is used, the union can be declared in a  
 41092 header file (which shall be included in the declarations section by using a **#include** construct  
 41093 within **%{** and **%}**), and a **typedef** used to define the symbol YYSTYPE to represent this union.  
 41094 The effect of **%union** is to provide the declaration of YYSTYPE directly from the *yacc* input.

41095 C-language declarations and definitions can appear in the declarations section, enclosed by the  
 41096 following marks:

41097 `%{ ... %}`

41098 These statements shall be copied into the code file, and have global scope within it so that they  
 41099 can be used in the rules and program sections.

41100 The application shall ensure that the declarations section is terminated by the token **%%**.

41101 **Grammar Rules in yacc**

41102 The rules section defines the context-free grammar to be accepted by the function *yacc* generates,  
 41103 and associates with those rules C-language actions and additional precedence information. The  
 41104 grammar is described below, and a formal definition follows.

41105 The rules section is comprised of one or more grammar rules. A grammar rule has the form:

41106 **A** : BODY ;

41107 The symbol **A** represents a non-terminal name, and **BODY** represents a sequence of zero or  
 41108 more *names*, *literals*, and *semantic actions* that can then be followed by optional *precedence rules*.  
 41109 Only the names and literals participate in the formation of the grammar; the semantic actions  
 41110 and precedence rules are used in other ways. The colon and the semicolon are *yacc* punctuation.  
 41111 If there are several successive grammar rules with the same left-hand side, the vertical bar ' | '   
 41112 can be used to avoid rewriting the left-hand side; in this case the semicolon appears only after  
 41113 the last rule. The BODY part can be empty (or empty of names and literals) to indicate that the  
 41114 non-terminal symbol matches the empty string.

41115 The *yacc* utility assigns a unique number to each rule. Rules using the vertical bar notation are  
 41116 distinct rules. The number assigned to the rule appears in the description file.

41117 The elements comprising a BODY are:

41118 *name, literal* These form the rules of the grammar: *name* is either a *token* or a *non-terminal*; *literal*  
 41119 stands for itself (less the lexically required quotation marks).

41120 *semantic action*

41121 With each grammar rule, the user can associate actions to be performed each time  
 41122 the rule is recognized in the input process. (Note that the word "action" can also  
 41123 refer to the actions of the parser—shift, reduce, and so on.)

41124 These actions can return values and can obtain the values returned by previous  
 41125 actions. These values are kept in objects of type YYSTYPE (see **%union**). The  
 41126 result value of the action shall be kept on the parse stack with the left-hand side of  
 41127 the rule, to be accessed by other reductions as part of their right-hand side. By  
 41128 using the *<tag>* information provided in the declarations section, the code  
 41129 generated by *yacc* can be strictly type checked and contain arbitrary information. In  
 41130 addition, the lexical analyzer can provide the same kinds of values for tokens, if  
 41131 desired.

41132 An action is an arbitrary C statement and as such can do input or output, call  
 41133 subprograms, and alter external variables. An action is one or more C statements  
 41134 enclosed in curly braces ' { ' and ' } '.

41135 Certain pseudo-variables can be used in the action. These are macros for access to  
 41136 data structures known internally to *yacc*.

41137 **\$\$** The value of the action can be set by assigning it to **\$\$**. If type  
 41138 checking is enabled and the type of the value to be assigned cannot  
 41139 be determined, a diagnostic message may be generated.

41140 **\$number** This refers to the value returned by the component specified by the  
 41141 token *number* in the right side of a rule, reading from left to right;  
 41142 *number* can be zero or negative. If *number* is zero or negative, it refers  
 41143 to the data associated with the name on the parser's stack preceding  
 41144 the leftmost symbol of the current rule. (That is, "\$0" refers to the  
 41145 name immediately preceding the leftmost name in the current rule to  
 41146 be found on the parser's stack and "\$-1" refers to the symbol to *its*



|       |                                  |                                                                                                               |
|-------|----------------------------------|---------------------------------------------------------------------------------------------------------------|
| 41147 |                                  | left.) If <i>number</i> refers to an element past the current point in the rule,                              |
| 41148 |                                  | or beyond the bottom of the stack, the result is undefined. If type                                           |
| 41149 |                                  | checking is enabled and the type of the value to be assigned cannot                                           |
| 41150 |                                  | be determined, a diagnostic message may be generated.                                                         |
| 41151 | <code>\$&lt;tag&gt;number</code> |                                                                                                               |
| 41152 |                                  | These correspond exactly to the corresponding symbols without the                                             |
| 41153 |                                  | <i>tag</i> inclusion, but allow for strict type checking (and preclude                                        |
| 41154 |                                  | unwanted type conversions). The effect is that the macro is expanded                                          |
| 41155 |                                  | to use <i>tag</i> to select an element from the YYSTYPE union (using                                          |
| 41156 |                                  | <i>dataname.tag</i> ). This is particularly useful if <i>number</i> is not positive.                          |
| 41157 | <code>\$&lt;tag&gt;\$</code>     |                                                                                                               |
| 41158 |                                  | This imposes on the reference the type of the union member                                                    |
| 41159 |                                  | referenced by <i>tag</i> . This construction is applicable when a reference                                   |
| 41160 |                                  | to a left context value occurs in the grammar, and provides yacc with                                         |
|       |                                  | a means for selecting a type.                                                                                 |
| 41161 |                                  | Actions can occur anywhere in a rule (not just at the end); an action can access                              |
| 41162 |                                  | values returned by actions to its left, and in turn the value it returns can be                               |
| 41163 |                                  | accessed by actions to its right. An action appearing in the middle of a rule shall be                        |
| 41164 |                                  | equivalent to replacing the action with a new non-terminal symbol and adding an                               |
| 41165 |                                  | empty rule with that non-terminal symbol on the left-hand side. The semantic                                  |
| 41166 |                                  | action associated with the new rule shall be equivalent to the original action. The                           |
| 41167 |                                  | use of actions within rules might introduce conflicts that would not otherwise                                |
| 41168 |                                  | exist.                                                                                                        |
| 41169 |                                  | By default, the value of a rule shall be the value of the first element in it. If the first                   |
| 41170 |                                  | element does not have a type (particularly in the case of a literal) and type                                 |
| 41171 |                                  | checking is turned on by <b>%type</b> , an error message shall result.                                        |
| 41172 | <i>precedence</i>                |                                                                                                               |
| 41173 |                                  | The keyword <b>%prec</b> can be used to change the precedence level associated with a                         |
| 41174 |                                  | particular grammar rule. Examples of this are in cases where a unary and binary                               |
| 41175 |                                  | operator have the same symbolic representation, but need to be given different                                |
| 41176 |                                  | precedences, or where the handling of an ambiguous if-else construction is                                    |
| 41177 |                                  | necessary. The reserved symbol <b>%prec</b> can appear immediately after the body of                          |
| 41178 |                                  | the grammar rule and can be followed by a token name or a literal. It shall cause                             |
| 41179 |                                  | the precedence of the grammar rule to become that of the following token name or                              |
|       |                                  | literal. The action for the rule as a whole can follow <b>%prec</b> .                                         |
| 41180 |                                  | If a program section follows, the application shall ensure that the grammar rules are terminated              |
| 41181 |                                  | by <b>%%</b> .                                                                                                |
| 41182 | <b>Programs Section</b>          |                                                                                                               |
| 41183 |                                  | The <i>programs</i> section can include the definition of the lexical analyzer <i>yylex()</i> , and any other |
| 41184 |                                  | functions; for example, those used in the actions specified in the grammar rules. It is unspecified           |
| 41185 |                                  | whether the programs section precedes or follows the semantic actions in the output file;                     |
| 41186 |                                  | therefore, if the application contains any macro definitions and declarations intended to apply to            |
| 41187 |                                  | the code in the semantic actions, it shall place them within " <b>%{ . . . %}</b> " in the declarations       |
| 41188 |                                  | section.                                                                                                      |

41189 **Input Grammar**

41190 The following input to *yacc* yields a parser for the input to *yacc*. This formal syntax takes  
 41191 precedence over the preceding text syntax description.

41192 The lexical structure is defined less precisely; **Lexical Structure of the Grammar** (on page 1066)  
 41193 defines most terms. The correspondence between the previous terms and the tokens below is as  
 41194 follows.

41195 **IDENTIFIER** This corresponds to the concept of *name*, given previously. It also includes  
 41196 literals as defined previously.

41197 **C\_IDENTIFIER** This is a name, and additionally it is known to be followed by a colon. A literal  
 41198 cannot yield this token.

41199 **NUMBER** A string of digits (a non-negative decimal integer).

41200 **TYPE, LEFT, MARK, LCURL, RCURL**

41201 These correspond directly to **%type**, **%left**, **%%**, **%{**, and **%}**.

41202 **{...}** This indicates C-language source code, with the possible inclusion of '\$'  
 41203 macros as discussed previously.

```

41204 /* Grammar for the input to yacc. */
41205 /* Basic entries. */
41206 /* The following are recognized by the lexical analyzer. */

41207 %token IDENTIFIER /* Includes identifiers and literals */
41208 %token C_IDENTIFIER /* identifier (but not literal)
41209 followed by a :: */
41210 %token NUMBER /* [0-9][0-9]* */

41211 /* Reserved words : %type=>TYPE %left=>LEFT, and so on */
41212 %token LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION

41213 %token MARK /* The %% mark. */
41214 %token LCURL /* The %{ mark. */
41215 %token RCURL /* The %} mark. */

41216 /* 8-bit character literals stand for themselves; */
41217 /* tokens have to be defined for multi-byte characters. */

41218 %start spec
41219 %%

41220 spec : defs MARK rules tail
41221 ;
41222 tail : MARK
41223 {
41224 /* In this action, set up the rest of the file. */
41225 }
41226 | /* Empty; the second MARK is optional. */
41227 ;
41228 defs : /* Empty. */
41229 | defs def
41230 ;
41231 def : START IDENTIFIER
41232 | UNION

```

```

41233 {
41234 /* Copy union definition to output. */
41235 }
41236 |
41237 {
41238 /* Copy C code to output file. */
41239 }
41240 RCURL
41241 |
41242 rword tag nlist
41243 rword : TOKEN
41244 |
41245 LEFT
41246 |
41247 RIGHT
41248 |
41249 NONASSOC
41250 |
41251 TYPE
41252 tag : /* Empty: union tag ID optional. */
41253 |
41254 '<' IDENTIFIER '>'
41255 nlist : nmno
41256 |
41257 nlist nmno
41258 nmno : IDENTIFIER /* Note: literal invalid with % type. */
41259 |
41260 IDENTIFIER NUMBER /* Note: invalid with % type. */
41261 ;
41262 /* Rule section */
41263 rules : C_IDENTIFIER rbody prec
41264 |
41265 rules rule
41266 rule : C_IDENTIFIER rbody prec
41267 |
41268 '|' rbody prec
41269 rbody : /* empty */
41270 |
41271 rbody IDENTIFIER
41272 |
41273 rbody act
41274 act : '{'
41275 {
41276 /* Copy action, translate $$, and so on. */
41277 }
41278 '}'
41279 prec : /* Empty */
41280 |
41281 PREC IDENTIFIER
41282 |
41283 PREC IDENTIFIER act
41284 |
41285 prec ';'
41286 ;

```

## Conflicts

The parser produced for an input grammar may contain states in which conflicts occur. The conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at least one LALR(1) conflict. The *yacc* utility shall resolve all conflicts, using either default rules or user-specified precedence rules.

Conflicts are either shift/reduce conflicts or reduce/reduce conflicts. A shift/reduce conflict is where, for a given state and lookahead symbol, both a shift action and a reduce action are possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions by two different rules are possible.

The rules below describe how to specify what actions to take when a conflict occurs. Not all shift/reduce conflicts can be successfully resolved this way because the conflict may be due to something other than ambiguity, so incautious use of these facilities can cause the language accepted by the parser to be much different from that which was intended. The description file shall contain sufficient information to understand the cause of the conflict. Where ambiguity is the reason either the default or explicit rules should be adequate to produce a working parser.

The declared precedences and associativities (see **Declarations Section** (on page 1066)) are used to resolve parsing conflicts as follows:

1. A precedence and associativity is associated with each grammar rule; it is the precedence and associativity of the last token or literal in the body of the rule. If the **%prec** keyword is used, it overrides this default. Some grammar rules might not have both precedence and associativity.
2. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have precedence and associativity associated with them, then the conflict is resolved in favor of the action (shift or reduce) associated with the higher precedence. If the precedences are the same, then the associativity is used; left associative implies reduce, right associative implies shift, and non-associative implies an error in the string being parsed.
3. When there is a shift/reduce conflict that cannot be resolved by rule 2, the shift is done. Conflicts resolved this way are counted in the diagnostic output described in **Error Handling**.
4. When there is a reduce/reduce conflict, a reduction is done by the grammar rule that occurs earlier in the input sequence. Conflicts resolved this way are counted in the diagnostic output described in **Error Handling**.

Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and reduce/reduce conflicts reported by *yacc* on either standard error or in the description file.

## Error Handling

The token **error** shall be reserved for error handling. The name **error** can be used in grammar rules. It indicates places where the parser can recover from a syntax error. The default value of **error** shall be 256. Its value can be changed using a **%token** declaration. The lexical analyzer should not return the value of **error**.

The parser shall detect a syntax error when it is in a state where the action associated with the lookahead symbol is **error**. A semantic action can cause the parser to initiate error handling by executing the macro YYERROR. When YYERROR is executed, the semantic action passes control back to the parser. YYERROR cannot be used outside of semantic actions.

When the parser detects a syntax error, it normally calls *yyerror()* with the character string "syntax error" as its argument. The call shall not be made if the parser is still recovering

41325 from a previous error when the error is detected. The parser is considered to be recovering from  
 41326 a previous error until the parser has shifted over at least three normal input symbols since the  
 41327 last error was detected or a semantic action has executed the macro *yterror*. The parser shall not  
 41328 call *yterror*() when YYERROR is executed.

41329 The macro function YYRECOVERING shall return 1 if a syntax error has been detected and the  
 41330 parser has not yet fully recovered from it. Otherwise, zero shall be returned.

41331 When a syntax error is detected by the parser, the parser shall check if a previous syntax error  
 41332 has been detected. If a previous error was detected, and if no normal input symbols have been  
 41333 shifted since the preceding error was detected, the parser checks if the lookahead symbol is an  
 41334 endmarker (see **Interface to the Lexical Analyzer**). If it is, the parser shall return with a non-  
 41335 zero value. Otherwise, the lookahead symbol shall be discarded and normal parsing shall  
 41336 resume.

41337 When YYERROR is executed or when the parser detects a syntax error and no previous error has  
 41338 been detected, or at least one normal input symbol has been shifted since the previous error was  
 41339 detected, the parser shall pop back one state at a time until the parse stack is empty or the  
 41340 current state allows a shift over **error**. If the parser empties the parse stack, it shall return with a  
 41341 non-zero value. Otherwise, it shall shift over **error** and then resume normal parsing. If the parser  
 41342 reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead  
 41343 symbol when parsing is resumed.

41344 The macro *yterror* in a semantic action shall cause the parser to act as if it has fully recovered  
 41345 from any previous errors. The macro *yyclearin* shall cause the parser to discard the current  
 41346 lookahead token. If the current lookahead token has not yet been read, *yyclearin* shall have no  
 41347 effect.

41348 The macro YYACCEPT shall cause the parser to return with the value zero. The macro  
 41349 YYABORT shall cause the parser to return with a non-zero value.

#### 41350 **Interface to the Lexical Analyzer**

41351 The *yylex*() function is an integer-valued function that returns a *token number* representing the  
 41352 kind of token read. If there is a value associated with the token returned by *yylex*() (see the  
 41353 discussion of *tag* above), it shall be assigned to the external variable *yylval*.

41354 If the parser and *yylex*() do not agree on these token numbers, reliable communication between  
 41355 them cannot occur. For (single-byte character) literals, the token is simply the numeric value of  
 41356 the character in the current character set. The numbers for other tokens can either be chosen by  
 41357 *yacc*, or chosen by the user. In either case, the **#define** construct of C is used to allow *yylex*() to  
 41358 return these numbers symbolically. The **#define** statements are put into the code file, and the  
 41359 header file if that file is requested. The set of characters permitted by *yacc* in an identifier is larger  
 41360 than that permitted by C. Token names found to contain such characters shall not be included in  
 41361 the **#define** declarations.

41362 If the token numbers are chosen by *yacc*, the tokens other than literals shall be assigned numbers  
 41363 greater than 256, although no order is implied. A token can be explicitly assigned a number by  
 41364 following its first appearance in the declarations section with a number. Names and literals not  
 41365 defined this way retain their default definition. All token numbers assigned by *yacc* shall be  
 41366 unique and distinct from the token numbers used for literals and user-assigned tokens. If  
 41367 duplicate token numbers cause conflicts in parser generation, *yacc* shall report an error;  
 41368 otherwise, it is unspecified whether the token assignment is accepted or an error is reported.

41369 The end of the input is marked by a special token called the *endmarker*, which has a token  
 41370 number that is zero or negative. (These values are invalid for any other token.) All lexical  
 41371 analyzers shall return zero or negative as a token number upon reaching the end of their input. If

the tokens up to, but excluding, the endmarker form a structure that matches the start symbol, the parser shall accept the input. If the endmarker is seen in any other context, it shall be considered an error.

### 41375 **Completing the Program**

In addition to *yyparse()* and *yylex()*, the functions *yyerror()* and *main()* are required to make a complete program. The application can supply *main()* and *yyerror()*, or those routines can be obtained from the *yacc* library.

### 41379 **Yacc Library**

The following functions shall appear only in the *yacc* library accessible through the **-ly** operand to *c99*; they can therefore be redefined by a conforming application:

#### 41382 **int main(void)**

This function shall call *yyparse()* and exit with an unspecified value. Other actions within this function are unspecified.

#### 41385 **int yyerror(const char \*s)**

This function shall write the NUL-terminated argument to standard error, followed by a `<newline>`.

The order of the **-ly** and **-ll** operands given to *c99* is significant; the application shall either provide its own *main()* function or ensure that **-ly** precedes **-ll**.

### 41390 **Debugging the Parser**

The parser generated by *yacc* shall have diagnostic facilities in it that can be optionally enabled at either compile time or at runtime (if enabled at compile time). The compilation of the runtime debugging code is under the control of *YYDEBUG*, a preprocessor symbol. If *YYDEBUG* has a non-zero value, the debugging code shall be included. If its value is zero, the code shall not be included.

In parsers where the debugging code has been included, the external **int yydebug** can be used to turn debugging on (with a non-zero value) and off (zero value) at runtime. The initial value of *yydebug* shall be zero.

When **-t** is specified, the code file shall be built such that, if *YYDEBUG* is not already defined at compilation time (using the *c99 -D YYDEBUG* option, for example), *YYDEBUG* shall be set explicitly to 1. When **-t** is not specified, the code file shall be built such that, if *YYDEBUG* is not already defined, it shall be set explicitly to zero.

The format of the debugging output is unspecified but includes at least enough information to determine the shift and reduce actions, and the input symbols. It also provides information about error recovery.

### 41406 **Algorithms**

The parser constructed by *yacc* implements an LALR(1) parsing algorithm as documented in the literature. It is unspecified whether the parser is table-driven or direct-coded.

A parser generated by *yacc* shall never request an input symbol from *yylex()* while in a state where the only actions other than the error action are reductions by a single rule.

The literature of parsing theory defines these concepts.

**Limits**

The *yacc* utility may have several internal tables. The minimum maximums for these tables are shown in the following table. The exact meaning of these values is implementation-defined. The implementation shall define the relationship between these values and between them and any error messages that the implementation may generate should it run out of space for any internal structure. An implementation may combine groups of these resources into a single pool as long as the total available to the user does not fall below the sum of the sizes specified by this section.

**Table 4-22** Internal Limits in *yacc*

| Limit      | Minimum Maximum | Description                                                                                                                                                                                                                                               |
|------------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {NTERMS}   | 126             | Number of tokens.                                                                                                                                                                                                                                         |
| {NNONTERM} | 200             | Number of non-terminals.                                                                                                                                                                                                                                  |
| {NPROD}    | 300             | Number of rules.                                                                                                                                                                                                                                          |
| {NSTATES}  | 600             | Number of states.                                                                                                                                                                                                                                         |
| {MEMSIZE}  | 5 200           | Length of rules. The total length, in names (tokens and non-terminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in <b>Grammar Rules in yacc</b> (on page 1068). |
| {ACTSIZE}  | 4 000           | Number of actions. “Actions” here (and in the description file) refer to parser actions (shift, reduce, and so on) not to semantic actions defined in <b>Grammar Rules in yacc</b> (on page 1068).                                                        |

**EXIT STATUS**

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

**CONSEQUENCES OF ERRORS**

If any errors are encountered, the run is aborted and *yacc* exits with a non-zero status. Partial code files and header files may be produced. The summary information in the description file shall always be produced if the **-v** flag is present.

**APPLICATION USAGE**

Historical implementations experience name conflicts on the names **yacc.tmp**, **yacc.acts**, **yacc.debug**, **y.tab.c**, **y.tab.h**, and **y.output** if more than one copy of *yacc* is running in a single directory at one time. The **-b** option was added to overcome this problem. The related problem of allowing multiple *yacc* parsers to be placed in the same file was addressed by adding a **-p** option to override the previously hard-coded **yy** variable prefix.

The description of the **-p** option specifies the minimal set of function and variable names that cause conflict when multiple parsers are linked together. **YYSTYPE** does not need to be changed. Instead, the programmer can use **-b** to give the header files for different parsers different names, and then the file with the **yylex()** for a given parser can include the header for that parser. Names such as **yyclearerr** do not need to be changed because they are used only in the actions; they do not have linkage. It is possible that an implementation has other names, either internal ones for implementing things such as **yyclearerr**, or providing non-standard features that it wants to change with **-p**.

Unary operators that are the same token as a binary operator in general need their precedence adjusted. This is handled by the **%prec** advisory symbol associated with the particular grammar rule defining that unary operator. (See **Grammar Rules in yacc** (on page 1068).) Applications are not required to use this operator for unary operators, but the grammars that do not require it are rare.

#### 41464 EXAMPLES

Access to the *yacc* library is obtained with library search operands to *c99*. To use the *yacc* library *main()*:

```
41467 c99 y.tab.c -l y
```

Both the *lex* library and the *yacc* library contain *main()*. To access the *yacc* *main()*:

```
41469 c99 y.tab.c lex.yy.c -l y -l l
```

This ensures that the *yacc* library is searched first, so that its *main()* is used.

The historical *yacc* libraries have contained two simple functions that are normally coded by the application programmer. These functions are similar to the following code:

```
41473 #include <locale.h>
41474 int main(void)
41475 {
41476 extern int yyparse();
41477 setlocale(LC_ALL, "");
41478 /* If the following parser is one created by lex, the
41479 application must be careful to ensure that LC_CTYPE
41480 and LC_COLLATE are set to the POSIX locale. */
41481 (void) yyparse();
41482 return (0);
41483 }
41484 #include <stdio.h>
41485 int yyerror(const char *msg)
41486 {
41487 (void) fprintf(stderr, "%s\n", msg);
41488 return (0);
41489 }
```

#### 41490 RATIONALE

The references in **Referenced Documents** (on page xxxi) may be helpful in constructing the parser generator. The referenced DeRemer and Pennello article (along with the works it references) describes a technique to generate parsers that conform to this volume of IEEE Std 1003.1-2001. Work in this area continues to be done, so implementors should consult current literature before doing any new implementations. The original Knuth article is the theoretical basis for this kind of parser, but the tables it generates are impractically large for reasonable grammars and should not be used. The “equivalent to” wording is intentional to assure that the best tables that are LALR(1) can be generated.

There has been confusion between the class of grammars, the algorithms needed to generate parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal. In particular, a parser generator that accepts the full range of LR(1) grammars need not generate a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars) for a grammar that happens to be SLR(1). Such an implementation need not recognize the case, either; table compression can yield the SLR(1) table (or one even smaller than that) without



41505 recognizing that the grammar is SLR(1). The speed of an LR(1) parser for any class is dependent  
 41506 more upon the table representation and compression (or the code generation if a direct parser is  
 41507 generated) than upon the class of grammar that the table generator handles.

41508 The speed of the parser generator is somewhat dependent upon the class of grammar it handles.  
 41509 However, the original Knuth article algorithms for constructing LR parsers were judged by its  
 41510 author to be impractically slow at that time. Although full LR is more complex than LALR(1), as  
 41511 computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock  
 41512 execution time) is becoming less significant.

41513 Potential authors are cautioned that the referenced DeRemer and Pennello article previously  
 41514 cited identifies a bug (an over-simplification of the computation of LALR(1) lookahead sets) in  
 41515 some of the LALR(1) algorithm statements that preceded it to publication. They should take the  
 41516 time to seek out that paper, as well as current relevant work, particularly Aho's.

41517 The **-b** option was added to provide a portable method for permitting yacc to work on multiple  
 41518 separate parsers in the same directory. If a directory contains more than one yacc grammar, and  
 41519 both grammars are constructed at the same time (by, for example, a parallel *make* program),  
 41520 conflict results. While the solution is not historical practice, it corrects a known deficiency in  
 41521 historical implementations. Corresponding changes were made to all sections that referenced  
 41522 the filenames **y.tab.c** (now "the code file"), **y.tab.h** (now "the header file"), and **y.output** (now  
 41523 "the description file").

41524 The grammar for yacc input is based on System V documentation. The textual description shows  
 41525 there that the ' ; ' is required at the end of the rule. The grammar and the implementation do not  
 41526 require this. (The use of **C\_IDENTIFIER** causes a reduce to occur in the right place.)

41527 Also, in that implementation, the constructs such as **%token** can be terminated by a semicolon,  
 41528 but this is not permitted by the grammar. The keywords such as **%token** can also appear in  
 41529 uppercase, which is again not discussed. In most places where '**%**' is used, '**\**' can be  
 41530 substituted, and there are alternate spellings for some of the symbols (for example, **%LEFT** can  
 41531 be "**%<**" or even "**\<**").

41532 Historically, **<tag>** can contain any characters except '**>**', including white space, in the  
 41533 implementation. However, since the *tag* must reference an ISO C standard union member, in  
 41534 practice conforming implementations need to support only the set of characters for ISO C  
 41535 standard identifiers in this context.

41536 Some historical implementations are known to accept actions that are terminated by a period.  
 41537 Historical implementations often allow '**\$**' in names. A conforming implementation does not  
 41538 need to support either of these behaviors.

41539 Deciding when to use **%prec** illustrates the difficulty in specifying the behavior of yacc. There  
 41540 may be situations in which the *grammar* is not, strictly speaking, in error, and yet yacc cannot  
 41541 interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances  
 41542 be resolved by providing additional information, such as using **%type** or **%union** declarations. It  
 41543 is often easier and it usually yields a smaller parser to take this alternative when it is  
 41544 appropriate.

41545 The size and execution time of a program produced without the runtime debugging code is  
 41546 usually smaller and slightly faster in historical implementations.

41547 Statistics messages from several historical implementations include the following types of  
 41548 information:

41549 *n*/512 terminals, *n*/300 non-terminals  
 41550 *n*/600 grammar rules, *n*/1500 states  
 41551 *n* shift/reduce, *n* reduce/reduce conflicts reported

41552  $n/350$  working sets used  
 41553 Memory: states, etc.  $n/15\,000$ , parser  $n/15\,000$   
 41554  $n/600$  distinct lookahead sets  
 41555  $n$  extra closures  
 41556  $n$  shift entries,  $n$  exceptions  
 41557  $n$  goto entries  
 41558  $n$  entries saved by goto default  
 41559 Optimizer space used: input  $n/15\,000$ , output  $n/15\,000$   
 41560  $n$  table entries,  $n$  zero  
 41561 Maximum spread:  $n$ , Maximum offset:  $n$

41562 The report of internal tables in the description file is left implementation-defined because all  
 41563 aspects of these limits are also implementation-defined. Some implementations may use  
 41564 dynamic allocation techniques and have no specific limit values to report.

41565 The format of the **y.output** file is not given because specification of the format was not seen to  
 41566 enhance applications portability. The listing is primarily intended to help human users  
 41567 understand and debug the parser; use of **y.output** by a conforming application script would be  
 41568 unusual. Furthermore, implementations have not produced consistent output and no popular  
 41569 format was apparent. The format selected by the implementation should be human-readable, in  
 41570 addition to the requirement that it be a text file.

41571 Standard error reports are not specifically described because they are seldom of use to  
 41572 conforming applications and there was no reason to restrict implementations.

41573 Some implementations recognize " $=\{$ " as equivalent to ' $\{$ ' because it appears in historical  
 41574 documentation. This construction was recognized and documented as obsolete as long ago as  
 41575 1978, in the referenced *Yacc: Yet Another Compiler-Compiler*. This volume of IEEE Std 1003.1-2001  
 41576 chose to leave it as obsolete and omit it.

41577 Multi-byte characters should be recognized by the lexical analyzer and returned as tokens. They  
 41578 should not be returned as multi-byte character literals. The token **error** that is used for error  
 41579 recovery is normally assigned the value 256 in the historical implementation. Thus, the token  
 41580 value 256, which is used in many multi-byte character sets, is not available for use as the value  
 41581 of a user-defined token.

41582 **FUTURE DIRECTIONS**  
 41583 None.

41584 **SEE ALSO**  
 41585 *c99*, *lex*

41586 **CHANGE HISTORY**  
 41587 First released in Issue 2.

41588 **Issue 5**  
 41589 The FUTURE DIRECTIONS section is added.

41590 **Issue 6**  
 41591 This utility is marked as part of the C-Language Development Utilities option.  
 41592 Minor changes have been added to align with the IEEE P1003.2b draft standard.  
 41593 The normative text is reworded to avoid use of the term “must” for application requirements.  
 41594 IEEE PASC Interpretation 1003.2 #177 is applied, changing the comment on **RCURL** from the }%  
 41595 token to the %}.

41596 **NAME**

41597           zcat — expand and concatenate data

41598 **SYNOPSIS**41599 xSI       zcat [*file...*]

41600

41601 **DESCRIPTION**

41602       The *zcat* utility shall write to standard output the uncompressed form of files that have been  
 41603       compressed using the *compress* utility. It is the equivalent of *uncompress -c*. Input files are not  
 41604       affected.

41605 **OPTIONS**

41606       None.

41607 **OPERANDS**

41608       The following operand shall be supported:

41609       *file*           The pathname of a file previously processed by the *compress* utility. If *file* already  
 41610       has the *.Z* suffix specified, it is used as submitted. Otherwise, the *.Z* suffix is  
 41611       appended to the filename prior to processing.

41612 **STDIN**41613       The standard input shall be used only if no *file* operands are specified, or if a *file* operand is *'-'*.41614 **INPUT FILES**41615       Input files shall be compressed files that are in the format produced by the *compress* utility.41616 **ENVIRONMENT VARIABLES**41617       The following environment variables shall affect the execution of *zcat*:

41618       *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 41619       (See the Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2,  
 41620       Internationalization Variables for the precedence of internationalization variables  
 41621       used to determine the values of locale categories.)

41622       *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 41623       internationalization variables.

41624       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 41625       characters (for example, single-byte as opposed to multi-byte characters in  
 41626       arguments).

41627       *LC\_MESSAGES* Determine the locale that should be used to affect the format and contents of  
 41628       diagnostic messages written to standard error.  
 41629

41630       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

41631 **ASYNCHRONOUS EVENTS**

41632       Default.

41633 **STDOUT**

41634       The compressed files given as input shall be written on standard output in their uncompressed  
 41635       form.

41636 **STDERR**

41637       The standard error shall be used only for diagnostic messages.

41638 **OUTPUT FILES**

41639           None.

41640 **EXTENDED DESCRIPTION**

41641           None.

41642 **EXIT STATUS**

41643           The following exit values shall be returned:

41644           0   Successful completion.

41645           &gt;0  An error occurred.

41646 **CONSEQUENCES OF ERRORS**

41647           Default.

41648 **APPLICATION USAGE**

41649           None.

41650 **EXAMPLES**

41651           None.

41652 **RATIONALE**

41653           None.

41654 **FUTURE DIRECTIONS**

41655           None.

41656 **SEE ALSO**41657           *compress, uncompress*41658 **CHANGE HISTORY**

41659           First released in Issue 4.

# Index

|                                           |             |                                       |            |
|-------------------------------------------|-------------|---------------------------------------|------------|
| <control>-V .....                         | 368         | string functions .....                | 167        |
| <control>-W .....                         | 368         | user-defined functions .....          | 169        |
| _CFLAGS .....                             | 215         | variables and special variables ..... | 160        |
| _LDFLAGS .....                            | 215         | background work .....                 |            |
| _LIBS .....                               | 215         | at .....                              | 144        |
| _POSIX_VDISABLE .....                     | 892         | batch .....                           | 190        |
| Account_Name .....                        | 108         | bg .....                              | 208        |
| actions equivalent to functions .....     | 7           | crontab .....                         | 275        |
| admin .....                               | <b>126</b>  | fg .....                              | 442        |
| ADV .....                                 | <b>9</b>    | jobs .....                            | 521        |
| AIO .....                                 | <b>9</b>    | nice .....                            | 664        |
| alias .....                               | 28, 48, 131 | nohup .....                           | 676        |
| alias substitution .....                  | 32          | renice .....                          | 819        |
| AND lists .....                           | 51          | BAR .....                             | <b>9</b>   |
| AND-OR list .....                         | 50          | basename .....                        | <b>187</b> |
| appending redirected output .....         | 44          | batch .....                           | <b>190</b> |
| ar .....                                  | <b>134</b>  | batch administration .....            | 104        |
| archives .....                            |             | batch authorization .....             | 103        |
| ar command .....                          | 134         | batch client-server interaction ..... | 101        |
| ARG_MAX .....                             | 1061        | batch environment .....               |            |
| arithmetic expansion .....                | 41          | services .....                        | 101        |
| arithmetic language .....                 |             | utilities .....                       | 101        |
| bc .....                                  | 193         | Batch Job Abort .....                 | 103, 114   |
| arithmetic precision and operations ..... | 7           | batch job creation .....              | 102        |
| array identifiers .....                   | 198         | Batch Job Execution .....             | 103, 106   |
| asa .....                                 | <b>141</b>  | Batch Job Exit .....                  | 103, 113   |
| asynchronous lists .....                  | 50          | batch job identifier .....            | 122        |
| at .....                                  | <b>144</b>  | Batch Job Message Request .....       | 116        |
| at-job .....                              | 144         | Batch Job Routing .....               | 102, 113   |
| automatic storage class .....             | 202         | batch job states .....                | 105        |
| awk .....                                 | <b>153</b>  | Batch Job Status Request .....        | 117        |
| actions .....                             | 164         | batch job tracking .....              | 102        |
| arithmetic functions .....                | 166         | batch notification .....              | 104        |
| escape sequences .....                    | 162         | batch queue .....                     | 101        |
| expression patterns .....                 | 164         | Batch Queue Status Request .....      | 119        |
| expressions .....                         | 156         | Batch Server Restart .....            | 114        |
| functions .....                           | 166         | batch services .....                  | 104        |
| grammar .....                             | 170         | bc .....                              | <b>193</b> |
| input/output and general functions .....  | 168         | grammar .....                         | 194        |
| lexical conventions .....                 | 176         | lexical conventions .....             | 196        |
| output statements .....                   | 165         | operations .....                      | 198        |
| overall program structure .....           | 155         | operators .....                       | 198        |
| pattern ranges .....                      | 164         | bcc (mailer blind carbon copy) .....  | 608        |
| patterns .....                            | 163         | BC_BASE_MAX .....                     | 18         |
| regular expressions .....                 | 161         | BC_DIM_MAX .....                      | 18         |
| special patterns .....                    | 163         | BC_SCALE_MAX .....                    | 18         |

|                                             |                |                                         |                          |
|---------------------------------------------|----------------|-----------------------------------------|--------------------------|
| BC_STRING_MAX.....                          | 18, 196        | uux.....                                | 980                      |
| BE.....                                     | <b>10</b>      | write.....                              | 1054                     |
| bg.....                                     | 28, 48, 208    | compilers.....                          | 211                      |
| binary primaries.....                       | 908            | c99.....                                | 464                      |
| break.....                                  | <b>65</b>      | fort77.....                             | 1063                     |
| built-in utilities.....                     | 28             | yacc.....                               | 52                       |
| builtin.....                                | 263            | compound commands.....                  | 50                       |
| c99.....                                    | <b>211</b>     | compound-list.....                      | <b>264</b>               |
| external symbols.....                       | 215            | compression.....                        | 264                      |
| standard libraries.....                     | 214            | compress.....                           | 951                      |
| cal.....                                    | <b>220</b>     | uncompress.....                         | 1079                     |
| can.....                                    | 1              | zcat.....                               | 3                        |
| carriage-control characters.....            | 141            | concurrent execution of processes.....  | 484                      |
| case conditional construct.....             | 53             | configuration values.....               | 139                      |
| cat.....                                    | <b>222</b>     | conforming application.....             | 46                       |
| cc (mailer carbon copy).....                | 608            | consequences of shell errors.....       | <b>69</b>                |
| CD.....                                     | <b>10</b>      | continue.....                           | 889                      |
| cd.....                                     | 28, 48, 226    | control characters.....                 | 3                        |
| cflow.....                                  | <b>231</b>     | controlling terminal.....               | 300                      |
| changing the current working directory..... | 7              | Coordinated Universal Time (UTC).....   | copy files commands..... |
| character counting.....                     | 1045           | cp.....                                 | 267                      |
| charmap.....                                |                | dd.....                                 | 302                      |
| with localedef.....                         | 558            | ln.....                                 | 549                      |
| writing names with locale.....              | 553            | mv.....                                 | 655                      |
| charmap file.....                           | 557, 892       | pax.....                                | 701                      |
| CHAR_BIT.....                               | 348            | cp.....                                 | <b>267</b>               |
| Checkpoint.....                             | 108            | cpio format.....                        | 721                      |
| checksums.....                              |                | CPT.....                                | <b>10</b>                |
| cksum.....                                  | 248            | CPU time.....                           | 917                      |
| chgrp.....                                  | <b>234</b>     | cron daemon.....                        | 278                      |
| chmod.....                                  | <b>237</b>     | crontab.....                            | <b>275</b>               |
| grammar.....                                | 240            | CS.....                                 | <b>10</b>                |
| chown.....                                  | <b>244</b>     | csplit.....                             | <b>279</b>               |
| cksum.....                                  | <b>248</b>     | ctags.....                              | <b>283</b>               |
| cmp.....                                    | <b>253</b>     | current working directory.....          | 3                        |
| codeset conversion.....                     | 506            | cut.....                                | <b>288</b>               |
| tr.....                                     | 926            | CX.....                                 | <b>10</b>                |
| COLL_WEIGHTS_MAX.....                       | 18             | cxref.....                              | <b>292</b>               |
| colon.....                                  | <b>67</b> , 69 | data keywords.....                      | 747                      |
| comm.....                                   | <b>256</b>     | date.....                               | <b>295</b>               |
| command.....                                | 28, 48, 259    | conversion specifications.....          | 295                      |
| command mode.....                           | 336            | modified conversion specifications..... | 296                      |
| command search and execution.....           | 48             | dd.....                                 | <b>302</b>               |
| command substitution.....                   | 40             | default queue.....                      | 118                      |
| communications commands.....                |                | deferred batch services.....            | 106                      |
| mailx.....                                  | 586            | Delete Batch Job Request.....           | 115                      |
| talk.....                                   | 900            | delta.....                              | <b>311</b>               |
| uucp.....                                   | 965            | destination.....                        | 123                      |
| uudecode.....                               | 969            | df.....                                 | <b>315</b>               |
| uuencode.....                               | 972            |                                         |                          |
| uustat.....                                 | 977            |                                         |                          |

|                                              |            |                                    |            |
|----------------------------------------------|------------|------------------------------------|------------|
| diff.....                                    | <b>319</b> | read command .....                 | 344        |
| binary output format .....                   | 321        | regular expressions .....          | 337        |
| default output format .....                  | 321        | shell escape command .....         | 346        |
| directory comparison format .....            | 320        | substitute command .....           | 344        |
| -c or -C output format .....                 | 322        | undo command .....                 | 345        |
| -e output format .....                       | 322        | write command .....                | 346        |
| -f output format .....                       | 322        | edit buffer .....                  | 355, 987   |
| directory commands                           |            | edit line .....                    | 855        |
| cd .....                                     | 226        | editors                            |            |
| pwd .....                                    | 758        | ed .....                           | 336        |
| directory lister .....                       | 571        | ex .....                           | 355        |
| dirname .....                                | <b>326</b> | sed .....                          | 841        |
| disk space commands                          |            | vi .....                           | 987        |
| df .....                                     | 315        | ED_FILE_MAX .....                  | 348        |
| du .....                                     | 329        | ED_LINE_MAX .....                  | 348        |
| ulimit .....                                 | 940        | effective group ID .....           | 3          |
| documentation .....                          | 631        | effective user ID .....            | 3          |
| dot .....                                    | <b>71</b>  | Eighth Edition UNIX .....          | 263        |
| double-quotes .....                          | 30         | env .....                          | <b>352</b> |
| du .....                                     | <b>329</b> | EPERM .....                        | 272        |
| duplicating an input file descriptor .....   | 45         | Error_Path .....                   | 109        |
| duplicating an output file descriptor .....  | 45         | escape character (backslash) ..... | 30         |
| echo .....                                   | <b>333</b> | escape sequences                   |            |
| ed .....                                     | <b>336</b> | awk .....                          | 162        |
| addresses .....                              | 338        | gencat .....                       | 474        |
| append command .....                         | 340        | lex .....                          | 540        |
| change command .....                         | 340        | establish the locale .....         | 7          |
| commands .....                               | 339        | eval .....                         | <b>73</b>  |
| copy command .....                           | 345        | ex .....                           | <b>355</b> |
| delete command .....                         | 341        | <backslash> .....                  | 367        |
| edit command .....                           | 341        | <control>-D command .....          | 390        |
| edit without checking command .....          | 341        | <newline> .....                    | 367        |
| filename command .....                       | 341        | abbreviate command .....           | 370        |
| global command .....                         | 341        | addressing .....                   | 361        |
| global non-matched command .....             | 346        | adjust window command .....        | 388        |
| help command .....                           | 342        | append command .....               | 371        |
| help-mode command .....                      | 342        | args command .....                 | 371        |
| insert command .....                         | 342        | autoindent option .....            | 392        |
| interactive global command .....             | 342        | autoprint option .....             | 393        |
| interactive global not-matched command ..... | 346        | autowrite option .....             | 393        |
| join command .....                           | 343        | beautify option .....              | 393        |
| line number command .....                    | 346        | change command .....               | 371        |
| list command .....                           | 343        | chdir command .....                | 372        |
| mark command .....                           | 343        | command descriptions .....         | 368        |
| move command .....                           | 343        | copy command .....                 | 372        |
| null command .....                           | 347        | delete command .....               | 372        |
| number command .....                         | 343        | directory option .....             | 393        |
| print command .....                          | 344        | edcompatible option .....          | 393        |
| prompt command .....                         | 344        | edit command .....                 | 372        |
| quit command .....                           | 344        | edit options .....                 | 392        |
| quit without checking command .....          | 344        | errorbells option .....            | 394        |

|                           |          |                                   |                                    |
|---------------------------|----------|-----------------------------------|------------------------------------|
| escape command.....       | 389      | tag command.....                  | 384                                |
| execute command .....     | 391      | taglength option .....            | 397                                |
| exrc command.....         | 394      | tags command.....                 | 398                                |
| file command.....         | 373      | term command.....                 | 398                                |
| global command.....       | 374      | terse command.....                | 398                                |
| ignorecase option .....   | 394      | unabbrev command.....             | 385                                |
| initialization.....       | 358      | undo command.....                 | 385                                |
| input editing.....        | 366      | unmap command .....               | 385                                |
| insert command.....       | 375      | version command.....              | 385                                |
| join command .....        | 375      | visual command .....              | 386                                |
| list command .....        | 376, 394 | warn command.....                 | 398                                |
| magic command .....       | 394      | window command .....              | 398                                |
| map command .....         | 376      | wrapmargin option.....            | 399                                |
| mark command.....         | 377      | wrapscan option.....              | 399                                |
| mesg command.....         | 394      | write command.....                | 386                                |
| move command.....         | 378      | write line number command .....   | 390                                |
| next command .....        | 378      | writeln option .....              | 399                                |
| number command.....       | 379      | xit command.....                  | 387                                |
| number option .....       | 395      | yank command .....                | 387                                |
| open command .....        | 379      | exec.....                         | <b>75</b> , 677                    |
| paragraphs option.....    | 395      | exec family .....                 | 28, 263, 662, 1061                 |
| preserve .....            | 355      | Execution_Time .....              | 110                                |
| preserve command .....    | 379      | EXINIT .....                      | 355                                |
| print command .....       | 380      | exit .....                        | <b>77</b>                          |
| prompt command .....      | 395      | exit status and errors.....       | 46                                 |
| put command.....          | 380      | exit status for commands.....     | 46                                 |
| quit command.....         | 380      | expand .....                      | <b>426</b>                         |
| read command .....        | 381      | export .....                      | <b>79</b>                          |
| readonly command.....     | 395      | expr.....                         | <b>429</b>                         |
| recover command.....      | 381      | matching expression.....          | 431                                |
| redraw command .....      | 396      | string operand.....               | 431                                |
| regular expressions ..... | 391      | expression argument .....         | 165                                |
| remap command .....       | 396      | expression list.....              | 165                                |
| replacement strings.....  | 391      | EXPR_NEST_MAX.....                | 18                                 |
| report command.....       | 396      | extended regular expression ..... | 153, 161, 271                      |
| rewind command .....      | 382      | .....                             | 455, 495, 539, 657, 710, 824, 1059 |
| scroll command.....       | 366, 396 | extension                         |                                    |
| sections command.....     | 396      | CX .....                          | 10                                 |
| set command .....         | 382      | OH .....                          | 13                                 |
| shell command.....        | 382      | XSI.....                          | 17                                 |
| shell option .....        | 397      | false .....                       | 28, 48, 434                        |
| shift left command .....  | 390      | fc .....                          | 28, 48, 436                        |
| shift right command ..... | 390      | FD.....                           | <b>10</b>                          |
| shiftwidth option.....    | 397      | fg .....                          | 28, 48, 442                        |
| showmatch option .....    | 397      | field splitting.....              | 42                                 |
| showmode command.....     | 397      | FIFO special files.....           | 641                                |
| slowopen command .....    | 397      | file .....                        | <b>444</b>                         |
| source command .....      | 383      | file access permissions.....      | 4                                  |
| substitute command .....  | 383      | file comparisons                  |                                    |
| suspend command.....      | 384      | cmp .....                         | 253                                |
| tabstop option .....      | 397      | comm .....                        | 256                                |



|                               |       |                                   |             |
|-------------------------------|-------|-----------------------------------|-------------|
| diff.....                     | 319   | more .....                        | 643         |
| uniq.....                     | 959   | nl .....                          | 668         |
| file contents .....           | 6     | paste .....                       | 687         |
| file conversion               |       | pax .....                         | 701         |
| cut .....                     | 288   | pr.....                           | 736         |
| dd .....                      | 302   | read.....                         | 816         |
| expand .....                  | 426   | sed.....                          | 841         |
| fold.....                     | 461   | tail.....                         | 897         |
| head.....                     | 503   | tee.....                          | 904         |
| join.....                     | 525   | tr.....                           | 926         |
| od .....                      | 679   | uncompress .....                  | 951         |
| paste .....                   | 687   | unexpand .....                    | 954         |
| patch.....                    | 691   | zcat .....                        | 1079        |
| sort.....                     | 870   | find.....                         | <b>452</b>  |
| strings .....                 | 879   | fold.....                         | <b>461</b>  |
| tail.....                     | 897   | for loop.....                     | 52          |
| tr.....                       | 926   | fort77 .....                      | <b>464</b>  |
| tsort .....                   | 934   | external symbols.....             | 467         |
| unexpand .....                | 954   | standard libraries.....           | 466         |
| uniq.....                     | 959   | FR.....                           | <b>10</b>   |
| uudecode.....                 | 969   | FSC.....                          | <b>10</b>   |
| uuencode.....                 | 972   | function definition command ..... | 54          |
| file creation.....            | 4     | function identifiers .....        | 198         |
| file descriptor.....          | 3, 43 | fuser.....                        | <b>470</b>  |
| file mode creation mask ..... | 3     | g-file.....                       | 311         |
| file permission commands      |       | gencat.....                       | <b>473</b>  |
| chgrp .....                   | 234   | escape sequences.....             | 474         |
| chmod .....                   | 237   | generated file .....              | 311         |
| chown .....                   | 244   | get .....                         | <b>476</b>  |
| umask.....                    | 942   | getconf .....                     | <b>484</b>  |
| file read .....               | 4     | getopts .....                     | 28, 48, 490 |
| file removal .....            | 6     | global storage class .....        | 202         |
| file searching                |       | GNU make .....                    | 627         |
| grep.....                     | 495   | grep.....                         | <b>495</b>  |
| file time values .....        | 6     | grouping commands.....            | 52          |
| file tree commands            |       | hash .....                        | <b>500</b>  |
| diff.....                     | 319   | head .....                        | <b>503</b>  |
| find.....                     | 452   | here-document .....               | 44          |
| ls.....                       | 571   | history command                   |             |
| mkdir.....                    | 638   | fc.....                           | 436         |
| rmdir .....                   | 830   | Hold Batch Job Request.....       | 116         |
| file write .....              | 4     | Hold_Types.....                   | 110         |
| filters                       |       | HOME.....                         | 372         |
| asa.....                      | 141   | hunk.....                         | 693         |
| awk .....                     | 153   | iconv.....                        | <b>506</b>  |
| compress .....                | 264   | id .....                          | <b>509</b>  |
| dd .....                      | 302   | if conditional construct.....     | 53          |
| expand .....                  | 426   | implementation-defined.....       | 1           |
| fold.....                     | 461   | inference rule .....              | 610         |
| head.....                     | 503   | input field separator.....        | 852         |
| iconv.....                    | 506   | input mode.....                   | 336         |

|                               |                                      |                                       |          |
|-------------------------------|--------------------------------------|---------------------------------------|----------|
| IP6 .....                     | 11                                   | delete aliases.....                   | 603      |
| ipcrm .....                   | 513                                  | delete messages .....                 | 597      |
| ipcs .....                    | 515                                  | delete messages and display.....      | 598      |
| I_ISVTX.....                  | 239                                  | direct messages to mbox.....          | 600      |
| jobs .....                    | 28, 48, 521                          | discard header fields.....            | 597      |
| Job_Owner .....               | 110                                  | display beginning of messages.....    | 603      |
| join .....                    | 525                                  | display current message number.....   | 604      |
| Join_Path .....               | 110                                  | display header summary .....          | 599      |
| Keep_Files .....              | 110                                  | display list of folders .....         | 599      |
| keyword-value pairs.....      | 123                                  | display message.....                  | 601      |
| kill.....                     | 28, 48, 530                          | display message size.....             | 602      |
| legacy.....                   | 1                                    | echo a string.....                    | 598      |
| lex.....                      | 535                                  | edit message.....                     | 598, 603 |
| actions .....                 | 541                                  | execute commands conditionally .....  | 600      |
| definitions .....             | 537                                  | exit .....                            | 598      |
| escape sequences.....         | 540                                  | follow up specified messages .....    | 599      |
| regular expressions .....     | 539                                  | help .....                            | 599      |
| rules.....                    | 538                                  | hold messages .....                   | 599      |
| table sizes .....             | 538                                  | internal variables.....               | 593      |
| user subroutines .....        | 539                                  | invoke a shell.....                   | 602      |
| lex, translation table.....   | 545                                  | invoke shell command .....            | 604      |
| libraries                     |                                      | list available commands .....         | 600      |
| ar command.....               | 134                                  | mail a message.....                   | 600      |
| LIMIT .....                   | 17                                   | null command .....                    | 604      |
| line counting .....           | 1045                                 | pipe message .....                    | 600      |
| LINE_MAX.....                 | 18, 154, 348-349, 356, 607, 849, 962 | process next specified message .....  | 600      |
| link .....                    | 547                                  | quit.....                             | 601      |
| lists.....                    | 50                                   | read mailx commands from a file ..... | 602      |
| AND-OR.....                   | 50                                   | receive mode .....                    | 586      |
| compound-list.....            | 50                                   | reply to a message.....               | 601      |
| ln .....                      | 549                                  | reply to a message list .....         | 601      |
| locale .....                  | 553                                  | retain header fields.....             | 602      |
| localedef.....                | 558                                  | save messages .....                   | 602      |
| Locate Batch Job Request..... | 117                                  | scroll header display.....            | 604      |
| locking file .....            | 241                                  | send mode.....                        | 586      |
| logger .....                  | 562                                  | set variables .....                   | 602      |
| logname .....                 | 564                                  | start-up .....                        | 593      |
| lp .....                      | 566                                  | touch messages.....                   | 603      |
| LR(1) grammars .....          | 1077                                 | undelete messages .....               | 603      |
| ls.....                       | 571                                  | unset variables .....                 | 603      |
| m4.....                       | 579                                  | write messages to a file .....        | 603      |
| macro processor.....          | 579                                  | Mail_Points .....                     | 111      |
| magic file .....              | 449                                  | Mail_Users .....                      | 111      |
| mailx.....                    | 586                                  | make.....                             | 610      |
| change current directory..... | 597                                  | default rules.....                    | 620      |
| change folder.....            | 598                                  | inference rules.....                  | 617      |
| command escapes .....         | 604                                  | internal macros .....                 | 619      |
| commands.....                 | 596                                  | libraries .....                       | 618      |
| copy messages .....           | 597                                  | macros.....                           | 616      |
| declare aliases .....         | 596                                  | makefile execution .....              | 614      |
| declare alternatives .....    | 597                                  | makefile syntax.....                  | 613      |

|                                          |            |                                                    |               |
|------------------------------------------|------------|----------------------------------------------------|---------------|
| target rules .....                       | 614        | MX.....                                            | <b>12</b>     |
| make, GNU version .....                  | 627        | NAME_MAX.....                                      | 139, 727      |
| man.....                                 | <b>631</b> | newgrp .....                                       | 28, 48, 660   |
| mathematical functions.....              | 9          | NGROUPS_MAX .....                                  | 663           |
| may .....                                | 2          | nice.....                                          | <b>664</b>    |
| MC1 .....                                | <b>11</b>  | Ninth Edition UNIX .....                           | 205, 335, 745 |
| MC2 .....                                | <b>11</b>  | nl .....                                           | <b>668</b>    |
| MC3 .....                                | <b>11</b>  | nm .....                                           | <b>672</b>    |
| mesg .....                               | <b>635</b> | noclobber option.....                              | 700           |
| message catalog generation .....         | 473        | nohup .....                                        | <b>676</b>    |
| MF.....                                  | <b>11</b>  | non-printable .....                                | 348, 848, 903 |
| MIL-STD-1753 .....                       | 468        | OB .....                                           | <b>12</b>     |
| Minimum_Cpu_Interval.....                | 108        | object files.....                                  | 672           |
| mkdir.....                               | <b>638</b> | od .....                                           | <b>679</b>    |
| mkfifo .....                             | <b>641</b> | OF.....                                            | <b>12</b>     |
| ML.....                                  | <b>11</b>  | OH .....                                           | <b>13</b>     |
| MLR.....                                 | <b>12</b>  | open file descriptors for reading and writing..... | 46            |
| Modify Batch Job Request .....           | 117        | open mode .....                                    | 355           |
| MON.....                                 | <b>12</b>  | option                                             |               |
| more.....                                | <b>643</b> | ADV.....                                           | 9             |
| discard and refresh.....                 | 649        | AIO .....                                          | 9             |
| display position .....                   | 652        | BAR.....                                           | 9             |
| examine new file.....                    | 651        | BE .....                                           | 10            |
| examine next file.....                   | 651        | CD.....                                            | 10            |
| examine previous file .....              | 651        | CPT .....                                          | 10            |
| go to beginning of file.....             | 649        | CS.....                                            | 10            |
| go to end-of-file.....                   | 649        | FD.....                                            | 10            |
| go to tag.....                           | 651        | FR.....                                            | 10            |
| help .....                               | 648        | FSC.....                                           | 10            |
| invoke editor .....                      | 651        | IP6.....                                           | 11            |
| mark position .....                      | 650        | MC1 .....                                          | 11            |
| quit.....                                | 652        | MC2 .....                                          | 11            |
| refresh the screen.....                  | 649        | MC3.....                                           | 11            |
| repeat search.....                       | 650        | MF.....                                            | 11            |
| repeat search in reverse.....            | 651        | ML.....                                            | 11            |
| return to mark.....                      | 650        | MLR.....                                           | 12            |
| return to previous position .....        | 650        | MON .....                                          | 12            |
| scroll backward one half screenful ..... | 649        | MPR.....                                           | 12            |
| scroll backward one line .....           | 648        | MSG.....                                           | 12            |
| scroll backward one screenful .....      | 648        | MX .....                                           | 12            |
| scroll forward one half screenful .....  | 649        | PIO .....                                          | 13            |
| scroll forward one line .....            | 648        | PS .....                                           | 13            |
| scroll forward one screenful .....       | 648        | RS .....                                           | 13            |
| search backward for pattern .....        | 650        | RTS .....                                          | 13            |
| search forward for pattern .....         | 650        | SD.....                                            | 13            |
| skip forward one line.....               | 649        | SEM .....                                          | 13            |
| motion command .....                     | 856        | SHM .....                                          | 13            |
| Move Batch Job Request .....             | 118        | SIO .....                                          | 14            |
| MPR.....                                 | <b>12</b>  | SPI.....                                           | 14            |
| MSG.....                                 | <b>12</b>  | SPN .....                                          | 14            |
| mv .....                                 | <b>655</b> | SS.....                                            | 14            |

|                                                |                    |                                         |             |
|------------------------------------------------|--------------------|-----------------------------------------|-------------|
| TCT.....                                       | 14                 | cpio file data.....                     | 723         |
| TEF.....                                       | 14                 | cpio filename.....                      | 723         |
| THR.....                                       | 14                 | cpio header.....                        | 721         |
| TMO.....                                       | 15                 | cpio interchange format.....            | 721         |
| TMR.....                                       | 15                 | cpio special entries.....               | 723         |
| TPI.....                                       | 15                 | extended header.....                    | 714         |
| TPP.....                                       | 15                 | extended header file times.....         | 717         |
| TPS.....                                       | 15                 | extended header keyword precedence..... | 717         |
| TRC.....                                       | 15                 | list mode format specifications.....    | 709         |
| TRI.....                                       | 15                 | ustar format.....                       | 717         |
| TRL.....                                       | 15                 | ustar interchange format.....           | 717         |
| TSA.....                                       | 16                 | PIO.....                                | <b>13</b>   |
| TSF.....                                       | 16                 | pipelines.....                          | 49          |
| TSH.....                                       | 16                 | portable character set.....             | 616         |
| TSP.....                                       | 16                 | positional parameters.....              | 33          |
| TSS.....                                       | 16                 | POSIX2_BC_BASE_MAX.....                 | 17-18       |
| TYM.....                                       | 16                 | POSIX2_BC_DIM_MAX.....                  | 17-18       |
| UP.....                                        | 16                 | POSIX2_BC_SCALE_MAX.....                | 17-18       |
| XSR.....                                       | 17                 | POSIX2_BC_STRING_MAX.....               | 17-18       |
| OR lists.....                                  | 51                 | POSIX2_COLL_WEIGHTS_MAX.....            | 17, 19      |
| ordinary identifiers.....                      | 198                | POSIX2_EXPR_NEST_MAX.....               | 17, 19      |
| Output_Path.....                               | 111                | POSIX2_LINE_MAX.....                    | 17, 19      |
| paginators.....                                |                    | POSIX2_RE_DUP_MAX.....                  | 17, 19      |
| more.....                                      | 643                | POSIX2_SYMLINKS.....                    | 19          |
| parameter expansion.....                       | 37                 | pr.....                                 | <b>736</b>  |
| parameters and variables.....                  | 33                 | print-related commands.....             |             |
| paste.....                                     | <b>687</b>         | fold.....                               | 461         |
| patch.....                                     | <b>691</b>         | lp.....                                 | 566         |
| filename determination.....                    | 694                | pr.....                                 | 736         |
| patch application.....                         | 694                | printf.....                             | <b>741</b>  |
| patch file format.....                         | 693                | Priority.....                           | 112         |
| pathchk.....                                   | <b>697</b>         | privileges.....                         | 636, 666    |
| pathname expansion.....                        | 42                 | process attributes.....                 | 3           |
| pathname manipulation.....                     |                    | process group ID.....                   | 3           |
| basename.....                                  | 187                | process ID.....                         | 3           |
| dirname.....                                   | 326                | process status report.....              | 751         |
| pathchk.....                                   | 697                | prs.....                                | <b>746</b>  |
| pathname resolution.....                       | 7                  | PS.....                                 | <b>13</b>   |
| PATH_MAX.....                                  | 18, 733, 826       | ps.....                                 | <b>751</b>  |
| pattern matching.....                          | 452, 710, 966, 982 | public locale.....                      | 553         |
| definition.....                                | 62                 | pwd.....                                | 28, 48, 758 |
| in case statements.....                        | 53                 | qalter.....                             | <b>760</b>  |
| in shell variables.....                        | 39                 | qdel.....                               | <b>769</b>  |
| pattern matching notation.....                 | 62, 458, 726       | qhold.....                              | <b>772</b>  |
| pattern scanning and processing language.....  |                    | qmove.....                              | <b>775</b>  |
| at.....                                        | 153                | qmsg.....                               | <b>778</b>  |
| patterns matching a single character.....      | 62                 | qrerun.....                             | <b>781</b>  |
| patterns matching multiple characters.....     | 63                 | qrls.....                               | <b>783</b>  |
| patterns used for filename expansion.....      | 63                 | qselect.....                            | <b>786</b>  |
| pax.....                                       | <b>701</b>         | qsig.....                               | <b>795</b>  |
| archive character set encoding / decoding..... | 732                | qstat.....                              | <b>798</b>  |

|                                    |                                        |                                                     |             |
|------------------------------------|----------------------------------------|-----------------------------------------------------|-------------|
| qsub .....                         | <b>803</b>                             | addresses .....                                     | 843         |
| Queue Batch Job Request .....      | 118                                    | editing commands .....                              | 843         |
| quote removal .....                | 42                                     | regular expressions .....                           | 843         |
| quoting .....                      | 30                                     | Select Batch Jobs Request .....                     | 120         |
| read .....                         | 28, 48, 816                            | SEM .....                                           | <b>13</b>   |
| readonly .....                     | <b>82</b>                              | sequential lists .....                              | 51          |
| real group ID .....                | 3                                      | Server Shutdown Request .....                       | 120         |
| real user ID .....                 | 3                                      | Server Status Request .....                         | 121         |
| redirecting input .....            | 44                                     | session membership .....                            | 3           |
| redirecting output .....           | 44                                     | set .....                                           | <b>86</b>   |
| redirection .....                  | 43                                     | set-group-ID .....                                  | 3, 273      |
| regular expressions .....          | 161, 271, 337, 361                     | set-user-ID .....                                   | 3, 242, 273 |
| .....                              | 391, 431, 455, 495, 539, 647, 657, 668 | set-user-ID scripts .....                           | 864         |
| .....                              | 707, 824, 843, 1004, 1007, 1059        | sh .....                                            | <b>850</b>  |
| related to shell patterns .....    | 62                                     | command history list .....                          | 854         |
| relational database operator ..... | 525                                    | command line editing .....                          | 854         |
| Release Batch Job Request .....    | 119                                    | vi line editing command mode .....                  | 856         |
| remove directories .....           | 830                                    | vi line editing insert mode .....                   | 855         |
| remove files .....                 | 823                                    | vi-mode command line editing .....                  | 855         |
| renice .....                       | <b>819</b>                             | shall .....                                         | 2           |
| requested batch services .....     | 115                                    | shell command language .....                        | 29          |
| Rerun Batch Job Request .....      | 120                                    | alias substitution .....                            | 32          |
| Rerunable .....                    | 112                                    | appending redirected output .....                   | 44          |
| reserved words .....               | 33                                     | arithmetic expansion .....                          | 41          |
| Resource_List .....                | 112                                    | command substitution .....                          | 40          |
| return .....                       | <b>84</b>                              | compound commands .....                             | 52          |
| RE_DUP_MAX .....                   | 18                                     | consequences of shell errors .....                  | 46          |
| rm .....                           | <b>823</b>                             | double-quotes .....                                 | 30          |
| rm del .....                       | <b>828</b>                             | duplicating an input file descriptor .....          | 45          |
| rm dir .....                       | <b>830</b>                             | duplicating an output file descriptor .....         | 45          |
| root directory .....               | 3                                      | escape character (backslash) .....                  | 30          |
| RS .....                           | <b>13</b>                              | exit status and errors .....                        | 46          |
| RTS .....                          | <b>13</b>                              | exit status for commands .....                      | 46          |
| sact .....                         | <b>833</b>                             | field splitting .....                               | 42          |
| saved set-group-ID .....           | 3                                      | function definition command .....                   | 54          |
| saved set-user-ID .....            | 3                                      | grammar .....                                       | 55          |
| sccs .....                         | <b>836</b>                             | here-document .....                                 | 44          |
| SCCS commands                      |                                        | introduction .....                                  | 29          |
| admin .....                        | 126                                    | lists .....                                         | 50          |
| delta .....                        | 311                                    | open file descriptors for reading and writing ..... | 46          |
| get .....                          | 476                                    | parameter expansion .....                           | 37          |
| prs .....                          | 746                                    | parameters and variables .....                      | 33          |
| rm del .....                       | 828                                    | pathname expansion .....                            | 42          |
| sact .....                         | 833                                    | pattern matching notation .....                     | 62          |
| sccs .....                         | 836                                    | patterns matching a single character .....          | 62          |
| unget .....                        | 957                                    | patterns matching multiple characters .....         | 63          |
| val .....                          | 984                                    | patterns used for filename expansion .....          | 63          |
| what .....                         | 1048                                   | pipelines .....                                     | 49          |
| SD .....                           | <b>13</b>                              | positional parameters .....                         | 33          |
| search pattern .....               | 283                                    | quote removal .....                                 | 42          |
| sed .....                          | <b>841</b>                             | quoting .....                                       | 30          |

|                                             |                                   |
|---------------------------------------------|-----------------------------------|
| redirecting input .....                     | 44                                |
| redirecting output .....                    | 44                                |
| redirection .....                           | 43                                |
| reserved words .....                        | 33                                |
| shell commands .....                        | 47                                |
| shell execution environment.....            | 61                                |
| shell grammar lexical conventions .....     | 55                                |
| shell grammar rules .....                   | 56                                |
| shell variables.....                        | 34                                |
| signals and error handling.....             | 61                                |
| simple commands .....                       | 47                                |
| single-quotes .....                         | 30                                |
| special built-in utilities.....             | 64                                |
| special parameters.....                     | 34                                |
| tilde expansion.....                        | 37                                |
| token recognition.....                      | 31                                |
| word expansions .....                       | 36                                |
| shell commands .....                        | 47                                |
| shell execution environment .....           | 61, 132, 818, 944                 |
| shell grammar.....                          | 55                                |
| shell grammar lexical conventions .....     | 55                                |
| shell grammar rules .....                   | 56                                |
| shell introduction.....                     | 29                                |
| shell variables.....                        | 34                                |
| Shell_Path_List.....                        | 112                               |
| shift.....                                  | <b>92</b>                         |
| SHM.....                                    | <b>13</b>                         |
| should.....                                 | 2                                 |
| SIGCONT .....                               | 358                               |
| SIGHUP .....                                | 337, 358, 987, 1028               |
| SIGINT .....                                | 313, 337, 357, 1040               |
| Signal Batch Job Request.....               | 121                               |
| signal processes .....                      | 530                               |
| signals and error handling.....             | 61                                |
| SIGQUIT .....                               | 337                               |
| SIGTERM.....                                | 358                               |
| simple commands .....                       | 47                                |
| single-quotes.....                          | 30                                |
| SIO .....                                   | <b>14</b>                         |
| sleep.....                                  | <b>867</b>                        |
| SLR(1) grammars.....                        | 1077                              |
| sort .....                                  | <b>870</b>                        |
| special built-in .....                      | 263, 666, 678, 759, 864, 917, 933 |
| special built-in utilities .....            | 64                                |
| break.....                                  | 65, 69                            |
| characteristics.....                        | 64                                |
| colon .....                                 | 67                                |
| dot.....                                    | 71                                |
| eval .....                                  | 73                                |
| exec.....                                   | 75                                |
| exit .....                                  | 77                                |
| export .....                                | 79                                |
| readonly.....                               | 82                                |
| return.....                                 | 84                                |
| set.....                                    | 86                                |
| shift.....                                  | 92                                |
| times .....                                 | 94                                |
| trap .....                                  | 96                                |
| unset .....                                 | 99                                |
| special parameters.....                     | 34                                |
| special targets.....                        | 615                               |
| SPI .....                                   | <b>14</b>                         |
| split .....                                 | <b>876</b>                        |
| split files .....                           |                                   |
| csplit .....                                | 279                               |
| split .....                                 | 876                               |
| SPN .....                                   | <b>14</b>                         |
| spoofing .....                              | 83                                |
| SS.....                                     | <b>14</b>                         |
| standard error.....                         | 43                                |
| standard input.....                         | 43                                |
| standard output .....                       | 43                                |
| strings.....                                | <b>879</b>                        |
| strip.....                                  | <b>882</b>                        |
| stty .....                                  | <b>884</b>                        |
| combination modes .....                     | 889                               |
| control modes.....                          | 884                               |
| input modes.....                            | 885                               |
| local modes .....                           | 887                               |
| output modes .....                          | 886                               |
| special control character assignments ..... | 888                               |
| st_gid .....                                | 139                               |
| st_mode .....                               | 139                               |
| st_mtime.....                               | 139                               |
| st_size.....                                | 139                               |
| st_uid.....                                 | 139                               |
| superuser .....                             | 440, 577, 725                     |
| supplementary group IDs.....                | 3                                 |
| system configuration values .....           | 484                               |
| system name.....                            | 948                               |
| tabs.....                                   | <b>893</b>                        |
| tag file creation.....                      | 283                               |
| tail .....                                  | <b>897</b>                        |
| talk .....                                  | <b>900</b>                        |
| tar format.....                             | 717                               |
| target queue.....                           | 118                               |
| target rule .....                           | 610                               |
| TCT .....                                   | <b>14</b>                         |
| tee.....                                    | <b>904</b>                        |
| TEF.....                                    | <b>14</b>                         |
| terminal characteristics .....              |                                   |
| stty .....                                  | 884                               |

|                               |             |                                |                    |
|-------------------------------|-------------|--------------------------------|--------------------|
| tabs .....                    | 893         | user identity                  |                    |
| tput .....                    | 923         | id .....                       | 509                |
| tty .....                     | 936         | logname .....                  | 564                |
| terminate processes .....     | 530         | newgrp .....                   | 660                |
| terminology .....             | <b>1</b>    | who .....                      | 1050               |
| test .....                    | <b>907</b>  | User_List .....                | 113                |
| THR .....                     | <b>14</b>   | ustar format .....             | 717                |
| tilde expansion .....         | 37          | utility option parsing .....   | 490                |
| time .....                    | <b>915</b>  | uucp .....                     | <b>965</b>         |
| times .....                   | <b>94</b>   | uudecode .....                 | <b>969</b>         |
| TMO .....                     | <b>15</b>   | uuencode .....                 | <b>972</b>         |
| TMPDIR .....                  | 705         | uustat .....                   | <b>977</b>         |
| TMR .....                     | <b>15</b>   | uux .....                      | <b>980</b>         |
| token recognition .....       | 31          | val .....                      | <b>984</b>         |
| touch .....                   | <b>919</b>  | Variable_List .....            | 113                |
| TPI .....                     | <b>15</b>   | vi .....                       | <b>987</b>         |
| TPP .....                     | <b>15</b>   | <ESC> .....                    | 1027               |
| TPS .....                     | <b>15</b>   | append .....                   | 1008               |
| tput .....                    | <b>923</b>  | change .....                   | 1009               |
| tr .....                      | <b>926</b>  | change to end-of-line .....    | 1009               |
| Track Batch Job Request ..... | 121         | clear and redisplay .....      | 995                |
| trap .....                    | <b>96</b>   | command descriptions .....     | 988                |
| TRC .....                     | <b>15</b>   | control-D .....                | 1024               |
| TRI .....                     | <b>15</b>   | control-H .....                | 1024               |
| TRL .....                     | <b>15</b>   | control-T .....                | 1026               |
| trojan horse .....            | 577         | control-U .....                | 1026               |
| true .....                    | 28, 48, 932 | control-V .....                | 1026               |
| TSA .....                     | <b>16</b>   | current and line above .....   | 1002               |
| TSF .....                     | <b>16</b>   | delete .....                   | 1010               |
| TSH .....                     | <b>16</b>   | delete character .....         | 1019-1020          |
| tsort .....                   | <b>934</b>  | delete to end-of-line .....    | 1010               |
| TSP .....                     | <b>16</b>   | display information .....      | 994                |
| TSS .....                     | <b>16</b>   | edit the alternate file .....  | 996                |
| tty .....                     | <b>936</b>  | enter ex mode .....            | 1016               |
| TYM .....                     | <b>16</b>   | execute .....                  | 1007               |
| type .....                    | <b>938</b>  | execute an ex command .....    | 1006               |
| ulimit .....                  | <b>940</b>  | exit .....                     | 1022               |
| umask .....                   | 28, 48, 942 | find character .....           | 1011               |
| unalias .....                 | 28, 48, 946 | find regular expression .....  | 1004               |
| uname .....                   | <b>948</b>  | Initialization .....           | 988                |
| unary primaries .....         | 908         | input mode commands .....      | 1022               |
| uncompress .....              | <b>951</b>  | insert .....                   | 1012-1013          |
| undefined .....               | 2           | insert empty line .....        | 1014-1015          |
| unexpand .....                | <b>954</b>  | join .....                     | 1013               |
| unset .....                   | <b>957</b>  | mark position .....            | 1013               |
| uniq .....                    | <b>959</b>  | move back .....                | 1002, 1008         |
| unlink .....                  | <b>963</b>  | move cursor .....              | 994, 997-998, 1017 |
| unset .....                   | <b>99</b>   | move down .....                | 995                |
| unspecified .....             | 2           | move forward .....             | 1002               |
| until loop .....              | 54          | move to bigword .....          | 1011, 1019         |
| UP .....                      | <b>16</b>   | move to bottom of screen ..... | 1013               |

|                                         |              |
|-----------------------------------------|--------------|
| move to first character in line .....   | 1005         |
| move to first non-<blank> .....         | 1001         |
| move to line .....                      | 1012         |
| move to matching character .....        | 998          |
| move to middle of screen .....          | 1014         |
| move to next section .....              | 1001         |
| move to specific column .....           | 1003         |
| move to top of screen .....             | 1012         |
| move to word .....                      | 1011, 1018   |
| move up .....                           | 995          |
| newline .....                           | 1025         |
| nul .....                               | 1024, 1027   |
| page backwards .....                    | 993          |
| page forward .....                      | 994          |
| put from buffer .....                   | 1015         |
| redraw screen .....                     | 996          |
| redraw window .....                     | 1021         |
| regular expression .....                | 1007         |
| repeat .....                            | 1003         |
| repeat find .....                       | 1006, 1014   |
| repeat substitution .....               | 999          |
| replace character .....                 | 1016-1017    |
| replace text with command .....         | 997          |
| return to previous context .....        | 999-1000     |
| return to previous section .....        | 1000         |
| reverse case .....                      | 1007         |
| reverse find character .....            | 1003         |
| scroll backward .....                   | 996          |
| scroll backward by line .....           | 996          |
| scroll forward .....                    | 993          |
| scroll forward by line .....            | 993          |
| search for tagstring .....              | 997          |
| shift left .....                        | 1006         |
| shift right .....                       | 1006         |
| substitute character .....              | 1017         |
| substitute lines .....                  | 1017         |
| terminate command or input mode .....   | 996          |
| undo .....                              | 1018         |
| undo current line .....                 | 1018         |
| yank .....                              | 1020         |
| yank current line .....                 | 1020         |
| visual mode .....                       | 355          |
| wait .....                              | 28, 48, 1041 |
| warning                                 |              |
| OB .....                                | 12           |
| OF .....                                | 12           |
| wc .....                                | <b>1045</b>  |
| what .....                              | <b>1048</b>  |
| while loop .....                        | 54           |
| who .....                               | <b>1050</b>  |
| word counting .....                     | 1045         |
| word expansions .....                   | 36           |
| write .....                             | <b>1054</b>  |
| xargs .....                             | <b>1057</b>  |
| XSI .....                               | <b>17</b>    |
| XSR .....                               | <b>17</b>    |
| yacc .....                              | <b>1063</b>  |
| algorithms .....                        | 1074         |
| code file .....                         | 1065         |
| completing the program .....            | 1074         |
| conflicts .....                         | 1072         |
| debugging the parser .....              | 1074         |
| declarations section .....              | 1066         |
| description file .....                  | 1065         |
| error handling .....                    | 1072         |
| grammar rules .....                     | 1068         |
| header file .....                       | 1065         |
| input grammar .....                     | 1070         |
| input language .....                    | 1065         |
| interface to the lexical analyzer ..... | 1073         |
| lexical structure of the grammar .....  | 1066         |
| library .....                           | 1074         |
| limits .....                            | 1075         |
| programs section .....                  | 1069         |
| zcat .....                              | <b>1079</b>  |



IEEE Std 1003.1™, 2003 Edition

The Open Group Technical Standard  
Base Specifications, Issue 6

Includes IEEE Std 1003.1™-2001 and IEEE Std 1003.1™-2001/Cor 1-2002

# Information Technology — Portable Operating System Interface (POSIX®)

## Rationale

Sponsor

**Portable Applications Standards Committee**  
of the  
**IEEE Computer Society**

and

**The Open Group**



THE *Open* GROUP

[This page intentionally left blank]

## Abstract

This standard is simultaneously ISO/IEC 9945:2002, IEEE Std 1003.1-2001, and forms the core of the Single UNIX Specification, Version 3.

The IEEE Std 1003.1, 2003 Edition includes IEEE Std 1003.1-2001/Cor 1-2002 incorporated into IEEE Std 1003.1-2001 (base document). The Corrigendum addresses problems discovered since the approval of IEEE Std 1003.1-2001. These changes are mainly due to resolving integration issues raised by the merger of the base documents that were incorporated into IEEE Std 1003.1-2001, which is the single common revision to IEEE Std 1003.1<sup>™</sup>-1996, IEEE Std 1003.2<sup>™</sup>-1992, ISO/IEC 9945-1:1996, ISO/IEC 9945-2:1993, and the Base Specifications of The Open Group Single UNIX<sup>®</sup> Specification, Version 2.

This standard defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. This standard is intended to be used by both applications developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of this standard:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

This standard describes the external characteristics and facilities that are of importance to applications developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

## Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, input/output (I/O), job control, network, portable operating system interface (POSIX<sup>®</sup>), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

Copyright © 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. All rights reserved. This printing is by the International Organization for Standardization with special permission of the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. Published in Switzerland.

#### Rationale (Informative)

Published 31 March 2003 by the Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, New York, NY 10016-5997, U.S.A.  
ISBN: 0-7381-3438-2 PDF 0-7381-3564-X / SS95078 CD-ROM 0-7381-3563-1 / SE95078  
Printed in the United States of America by the IEEE.

Published 31 March 2003 by The Open Group  
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, U.K.  
Document Number: C034  
ISBN: 1-931624-26-7  
Printed in the U.K. by The Open Group.

All rights reserved. No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission from both the IEEE and The Open Group.

Portions of this standard are derived with permission from copyrighted material owned by Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc.

#### Permissions

Authorization to photocopy portions of this standard for internal or personal use is granted provided that the appropriate fee is paid to the Copyright Clearance Center or the equivalent body outside of the U.S. Permission to make multiple copies for educational purposes in the U.S. requires agreement and a license fee to be paid to the Copyright Clearance Center.

Beyond these provisions, permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests to the copyright holders should be sent by email to [austin-group-permissions@opengroup.org](mailto:austin-group-permissions@opengroup.org).

#### Feedback

This standard has been prepared by the Austin Group. Feedback relating to the material contained in this standard may be submitted using the Austin Group web site at <http://www.opengroup.org/austin/defectform.html>.

## IEEE

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property, or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

**The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “AS IS”.**

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with the IEEE.<sup>1</sup> Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, U.S.A.

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE Standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

A patent holder has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and non-discriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Authorization to photocopy portions of any individual standard for internal or personal use is granted in the U.S. by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to the Copyright Clearance Center.<sup>2</sup> Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center. To arrange for payment of the licensing fee, please contact:

Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923, U.S.A., Tel.: +1 978 750 8400

Amendments, corrigenda, and interpretations for this standard, or information about the IEEE standards development process, may be found at <http://standards.ieee.org>.

Full catalog and ordering information on all IEEE publications is available from the IEEE Online Catalog & Store at <http://shop.ieee.org/store>.

1. For this standard, please send comments via the Austin Group as requested on page iii.

2. Please refer to the special provisions for this standard on page iii concerning permissions from both copyright holders and arrangements to cover photocopying and reproduction across the world, as well as by commercial organizations wishing to license the material for use in product documentation.

## The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The Open Group's mission is to offer all organizations concerned with open information infrastructures a forum to share knowledge, integrate open initiatives, and certify approved products and processes in a manner in which they continue to trust our impartiality.

In the global eCommerce world of today, no single economic entity can achieve independence while still ensuring interoperability. The assurance that products will interoperate with each other across differing systems and platforms is essential to the success of eCommerce and business workflow. The Open Group, with its proven testing and certification program, is the international guarantor of interoperability in the new century.

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group's members include some of the largest and most influential organizations in the world. The flexible structure of The Open Groups membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information is available on The Open Group web site at <http://www.opengroup.org>.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes the *Westwood* family of tests for this standard and the associated certification program for Version 3 of the Single UNIX Specification, as well tests for CDE, CORBA, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at <http://www.opengroup.org/testing>.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at <http://www.opengroup.org/pubs>.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at <http://www.opengroup.org/corrigenda>.

Full catalog and ordering information on all Open Group publications is available at <http://www.opengroup.org/pubs>.

# Contents

|                 |          |                                                                     |          |
|-----------------|----------|---------------------------------------------------------------------|----------|
| <b>Part</b>     | <b>A</b> | <b>Base Definitions.....</b>                                        | <b>1</b> |
| <b>Appendix</b> | <b>A</b> | <b>Rationale for Base Definitions.....</b>                          | <b>3</b> |
|                 | A.1      | Introduction .....                                                  | 3        |
|                 | A.1.1    | Scope.....                                                          | 3        |
|                 | A.1.2    | Conformance .....                                                   | 5        |
|                 | A.1.3    | Normative References .....                                          | 5        |
|                 | A.1.4    | Terminology .....                                                   | 5        |
|                 | A.1.5    | Portability .....                                                   | 8        |
|                 | A.1.5.1  | Codes .....                                                         | 8        |
|                 | A.1.5.2  | Margin Code Notation.....                                           | 8        |
|                 | A.2      | Conformance .....                                                   | 9        |
|                 | A.2.1    | Implementation Conformance.....                                     | 9        |
|                 | A.2.1.1  | Requirements.....                                                   | 9        |
|                 | A.2.1.2  | Documentation.....                                                  | 9        |
|                 | A.2.1.3  | POSIX Conformance .....                                             | 10       |
|                 | A.2.1.4  | XSI Conformance .....                                               | 10       |
|                 | A.2.1.5  | Option Groups.....                                                  | 11       |
|                 | A.2.1.6  | Options.....                                                        | 12       |
|                 | A.2.2    | Application Conformance.....                                        | 12       |
|                 | A.2.2.1  | Strictly Conforming POSIX Application.....                          | 12       |
|                 | A.2.2.2  | Conforming POSIX Application.....                                   | 12       |
|                 | A.2.2.3  | Conforming POSIX Application Using Extensions.....                  | 12       |
|                 | A.2.2.4  | Strictly Conforming XSI Application .....                           | 12       |
|                 | A.2.2.5  | Conforming XSI Application Using Extensions.....                    | 12       |
|                 | A.2.3    | Language-Dependent Services for the C Programming<br>Language ..... | 13       |
|                 | A.2.4    | Other Language-Related Specifications.....                          | 13       |
|                 | A.3      | Definitions .....                                                   | 13       |
|                 | A.4      | General Concepts.....                                               | 33       |
|                 | A.4.1    | Concurrent Execution .....                                          | 33       |
|                 | A.4.2    | Directory Protection .....                                          | 33       |
|                 | A.4.3    | Extended Security Controls.....                                     | 33       |
|                 | A.4.4    | File Access Permissions .....                                       | 33       |
|                 | A.4.5    | File Hierarchy .....                                                | 33       |
|                 | A.4.6    | Filenames.....                                                      | 34       |
|                 | A.4.7    | File Times Update.....                                              | 35       |
|                 | A.4.8    | Host and Network Byte Order.....                                    | 35       |
|                 | A.4.9    | Measurement of Execution Time.....                                  | 35       |
|                 | A.4.10   | Memory Synchronization.....                                         | 36       |
|                 | A.4.11   | Pathname Resolution .....                                           | 37       |
|                 | A.4.12   | Process ID Reuse .....                                              | 39       |

|         |                                                                |    |
|---------|----------------------------------------------------------------|----|
| A.4.13  | Scheduling Policy.....                                         | 39 |
| A.4.14  | Seconds Since the Epoch .....                                  | 39 |
| A.4.15  | Semaphore.....                                                 | 40 |
| A.4.16  | Thread-Safety.....                                             | 40 |
| A.4.17  | Tracing.....                                                   | 40 |
| A.4.18  | Treatment of Error Conditions for Mathematical Functions ..... | 40 |
| A.4.19  | Treatment of NaN Arguments for Mathematical Functions .....    | 40 |
| A.4.20  | Utility.....                                                   | 40 |
| A.4.21  | Variable Assignment .....                                      | 41 |
| A.5     | File Format Notation.....                                      | 41 |
| A.6     | Character Set.....                                             | 41 |
| A.6.1   | Portable Character Set.....                                    | 41 |
| A.6.2   | Character Encoding.....                                        | 42 |
| A.6.3   | C Language Wide-Character Codes .....                          | 42 |
| A.6.4   | Character Set Description File.....                            | 42 |
| A.6.4.1 | State-Dependent Character Encodings .....                      | 42 |
| A.7     | Locale.....                                                    | 44 |
| A.7.1   | General.....                                                   | 44 |
| A.7.2   | POSIX Locale .....                                             | 45 |
| A.7.3   | Locale Definition .....                                        | 45 |
| A.7.3.1 | LC_CTYPE.....                                                  | 46 |
| A.7.3.2 | LC_COLLATE.....                                                | 47 |
| A.7.3.3 | LC_MONETARY.....                                               | 49 |
| A.7.3.4 | LC_NUMERIC.....                                                | 50 |
| A.7.3.5 | LC_TIME.....                                                   | 50 |
| A.7.3.6 | LC_MESSAGES.....                                               | 51 |
| A.7.4   | Locale Definition Grammar.....                                 | 51 |
| A.7.4.1 | Locale Lexical Conventions.....                                | 51 |
| A.7.4.2 | Locale Grammar.....                                            | 51 |
| A.7.5   | Locale Definition Example.....                                 | 52 |
| A.8     | Environment Variables .....                                    | 55 |
| A.8.1   | Environment Variable Definition .....                          | 55 |
| A.8.2   | Internationalization Variables.....                            | 55 |
| A.8.3   | Other Environment Variables.....                               | 56 |
| A.9     | Regular Expressions .....                                      | 58 |
| A.9.1   | Regular Expression Definitions .....                           | 58 |
| A.9.2   | Regular Expression General Requirements.....                   | 59 |
| A.9.3   | Basic Regular Expressions .....                                | 60 |
| A.9.3.1 | BREs Matching a Single Character or Collating Element.....     | 60 |
| A.9.3.2 | BRE Ordinary Characters.....                                   | 60 |
| A.9.3.3 | BRE Special Characters.....                                    | 60 |
| A.9.3.4 | Periods in BREs.....                                           | 60 |
| A.9.3.5 | RE Bracket Expression .....                                    | 60 |
| A.9.3.6 | BREs Matching Multiple Characters.....                         | 62 |
| A.9.3.7 | BRE Precedence.....                                            | 62 |
| A.9.3.8 | BRE Expression Anchoring.....                                  | 62 |
| A.9.4   | Extended Regular Expressions .....                             | 63 |
| A.9.4.1 | EREs Matching a Single Character or Collating Element.....     | 63 |



|           |                                          |    |
|-----------|------------------------------------------|----|
| A.9.4.2   | ERE Ordinary Characters.....             | 63 |
| A.9.4.3   | ERE Special Characters.....              | 63 |
| A.9.4.4   | Periods in EREs.....                     | 63 |
| A.9.4.5   | ERE Bracket Expression.....              | 63 |
| A.9.4.6   | EREs Matching Multiple Characters.....   | 63 |
| A.9.4.7   | ERE Alternation.....                     | 63 |
| A.9.4.8   | ERE Precedence.....                      | 64 |
| A.9.4.9   | ERE Expression Anchoring.....            | 64 |
| A.9.5     | Regular Expression Grammar.....          | 64 |
| A.9.5.1   | BRE/ERE Grammar Lexical Conventions..... | 64 |
| A.9.5.2   | RE and Bracket Expression Grammar.....   | 64 |
| A.9.5.3   | ERE Grammar.....                         | 64 |
| A.10      | Directory Structure and Devices .....    | 65 |
| A.10.1    | Directory Structure and Files .....      | 65 |
| A.10.2    | Output Devices and Terminal Types .....  | 65 |
| A.11      | General Terminal Interface .....         | 65 |
| A.11.1    | Interface Characteristics.....           | 66 |
| A.11.1.1  | Opening a Terminal Device File .....     | 66 |
| A.11.1.2  | Process Groups.....                      | 66 |
| A.11.1.3  | The Controlling Terminal.....            | 67 |
| A.11.1.4  | Terminal Access Control .....            | 67 |
| A.11.1.5  | Input Processing and Reading Data.....   | 68 |
| A.11.1.6  | Canonical Mode Input Processing .....    | 68 |
| A.11.1.7  | Non-Canonical Mode Input Processing..... | 69 |
| A.11.1.8  | Writing Data and Output Processing ..... | 69 |
| A.11.1.9  | Special Characters.....                  | 69 |
| A.11.1.10 | Modem Disconnect.....                    | 69 |
| A.11.1.11 | Closing a Terminal Device File .....     | 69 |
| A.11.2    | Parameters that Can be Set .....         | 70 |
| A.11.2.1  | The termios Structure .....              | 70 |
| A.11.2.2  | Input Modes.....                         | 70 |
| A.11.2.3  | Output Modes .....                       | 70 |
| A.11.2.4  | Control Modes.....                       | 70 |
| A.11.2.5  | Local Modes .....                        | 70 |
| A.11.2.6  | Special Control Characters .....         | 71 |
| A.12      | Utility Conventions.....                 | 71 |
| A.12.1    | Utility Argument Syntax.....             | 71 |
| A.12.2    | Utility Syntax Guidelines .....          | 72 |
| A.13      | Headers .....                            | 74 |
| A.13.1    | Format of Entries.....                   | 74 |

|                 |          |                                                                |           |
|-----------------|----------|----------------------------------------------------------------|-----------|
| <b>Part</b>     | <b>B</b> | <b>System Interfaces.....</b>                                  | <b>75</b> |
| <b>Appendix</b> | <b>B</b> | <b>Rationale for System Interfaces.....</b>                    | <b>77</b> |
|                 | B.1      | Introduction .....                                             | 77        |
|                 | B.1.1    | Scope.....                                                     | 77        |
|                 | B.1.2    | Conformance .....                                              | 77        |
|                 | B.1.3    | Normative References .....                                     | 77        |
|                 | B.1.4    | Change History .....                                           | 77        |
|                 | B.1.5    | Terminology .....                                              | 83        |
|                 | B.1.6    | Definitions .....                                              | 83        |
|                 | B.1.7    | Relationship to Other Formal Standards .....                   | 83        |
|                 | B.1.8    | Portability .....                                              | 83        |
|                 | B.1.8.1  | Codes .....                                                    | 83        |
|                 | B.1.9    | Format of Entries.....                                         | 83        |
|                 | B.2      | General Information.....                                       | 84        |
|                 | B.2.1    | Use and Implementation of Functions .....                      | 84        |
|                 | B.2.2    | The Compilation Environment .....                              | 85        |
|                 | B.2.2.1  | POSIX.1 Symbols .....                                          | 85        |
|                 | B.2.2.2  | The Name Space.....                                            | 86        |
|                 | B.2.3    | Error Numbers.....                                             | 89        |
|                 | B.2.3.1  | Additional Error Numbers.....                                  | 93        |
|                 | B.2.4    | Signal Concepts.....                                           | 93        |
|                 | B.2.4.1  | Signal Generation and Delivery.....                            | 95        |
|                 | B.2.4.2  | Realtime Signal Generation and Delivery .....                  | 96        |
|                 | B.2.4.3  | Signal Actions .....                                           | 99        |
|                 | B.2.4.4  | Signal Effects on Other Functions .....                        | 102       |
|                 | B.2.5    | Standard I/O Streams.....                                      | 103       |
|                 | B.2.5.1  | Interaction of File Descriptors and Standard I/O Streams ..... | 103       |
|                 | B.2.5.2  | Stream Orientation and Encoding Rules .....                    | 103       |
|                 | B.2.6    | STREAMS .....                                                  | 103       |
|                 | B.2.6.1  | Accessing STREAMS.....                                         | 103       |
|                 | B.2.7    | XSI Interprocess Communication .....                           | 103       |
|                 | B.2.7.1  | IPC General Information.....                                   | 104       |
|                 | B.2.8    | Realtime .....                                                 | 104       |
|                 | B.2.8.1  | Realtime Signals .....                                         | 110       |
|                 | B.2.8.2  | Asynchronous I/O .....                                         | 112       |
|                 | B.2.8.3  | Memory Management .....                                        | 114       |
|                 | B.2.8.4  | Process Scheduling .....                                       | 127       |
|                 | B.2.8.5  | Clocks and Timers .....                                        | 134       |
|                 | B.2.9    | Threads.....                                                   | 150       |
|                 | B.2.9.1  | Thread-Safety.....                                             | 163       |
|                 | B.2.9.2  | Thread IDs .....                                               | 166       |
|                 | B.2.9.3  | Thread Mutexes.....                                            | 167       |
|                 | B.2.9.4  | Thread Scheduling.....                                         | 167       |
|                 | B.2.9.5  | Thread Cancellation .....                                      | 171       |
|                 | B.2.9.6  | Thread Read-Write Locks.....                                   | 175       |
|                 | B.2.9.7  | Thread Interactions with Regular File Operations .....         | 177       |
|                 | B.2.10   | Sockets.....                                                   | 177       |

|                 |                                                           |            |
|-----------------|-----------------------------------------------------------|------------|
| B.2.10.1        | Address Families.....                                     | 177        |
| B.2.10.2        | Addressing .....                                          | 177        |
| B.2.10.3        | Protocols .....                                           | 177        |
| B.2.10.4        | Routing.....                                              | 177        |
| B.2.10.5        | Interfaces.....                                           | 177        |
| B.2.10.6        | Socket Types.....                                         | 177        |
| B.2.10.7        | Socket I/O Mode.....                                      | 177        |
| B.2.10.8        | Socket Owner.....                                         | 178        |
| B.2.10.9        | Socket Queue Limits .....                                 | 178        |
| B.2.10.10       | Pending Error.....                                        | 178        |
| B.2.10.11       | Socket Receive Queue.....                                 | 178        |
| B.2.10.12       | Socket Out-of-Band Data State .....                       | 178        |
| B.2.10.13       | Connection Indication Queue .....                         | 178        |
| B.2.10.14       | Signals .....                                             | 178        |
| B.2.10.15       | Asynchronous Errors .....                                 | 178        |
| B.2.10.16       | Use of Options.....                                       | 178        |
| B.2.10.17       | Use of Sockets for Local UNIX Connections.....            | 178        |
| B.2.10.18       | Use of Sockets over Internet Protocols.....               | 178        |
| B.2.10.19       | Use of Sockets over Internet Protocols Based on IPv4..... | 178        |
| B.2.10.20       | Use of Sockets over Internet Protocols Based on IPv6..... | 178        |
| B.2.11          | Tracing.....                                              | 179        |
| B.2.11.1        | Objectives .....                                          | 179        |
| B.2.11.2        | Trace Model.....                                          | 184        |
| B.2.11.3        | Trace Programming Examples .....                          | 189        |
| B.2.11.4        | Rationale on Trace for Debugging.....                     | 197        |
| B.2.11.5        | Rationale on Trace Event Type Name Space.....             | 197        |
| B.2.11.6        | Rationale on Trace Events Type Filtering .....            | 199        |
| B.2.11.7        | Tracing, pthread API.....                                 | 201        |
| B.2.11.8        | Rationale on Triggering .....                             | 202        |
| B.2.11.9        | Rationale on Timestamp Clock.....                         | 202        |
| B.2.11.10       | Rationale on Different Overrun Conditions.....            | 203        |
| B.2.12          | Data Types.....                                           | 203        |
| B.3             | System Interfaces .....                                   | 206        |
| B.3.1           | Examples for Spawn.....                                   | 206        |
| <b>Part</b>     | <b>C Shell and Utilities.....</b>                         | <b>217</b> |
| <b>Appendix</b> | <b>C Rationale for Shell and Utilities .....</b>          | <b>219</b> |
| C.1             | Introduction .....                                        | 219        |
| C.1.1           | Scope.....                                                | 219        |
| C.1.2           | Conformance .....                                         | 219        |
| C.1.3           | Normative References .....                                | 219        |
| C.1.4           | Change History .....                                      | 219        |
| C.1.5           | Terminology .....                                         | 220        |
| C.1.6           | Definitions.....                                          | 220        |
| C.1.7           | Relationship to Other Documents.....                      | 220        |
| C.1.7.1         | System Interfaces.....                                    | 220        |
| C.1.7.2         | Concepts Derived from the ISO C Standard.....             | 221        |

|          |                                                                       |     |
|----------|-----------------------------------------------------------------------|-----|
| C.1.8    | Portability .....                                                     | 221 |
| C.1.8.1  | Codes .....                                                           | 221 |
| C.1.9    | Utility Limits .....                                                  | 222 |
| C.1.10   | Grammar Conventions .....                                             | 225 |
| C.1.11   | Utility Description Defaults .....                                    | 225 |
| C.1.12   | Considerations for Utilities in Support of Files of Arbitrary Size .. | 228 |
| C.1.13   | Built-In Utilities .....                                              | 229 |
| C.2      | Shell Command Language .....                                          | 231 |
| C.2.1    | Shell Introduction .....                                              | 231 |
| C.2.2    | Quoting .....                                                         | 231 |
| C.2.2.1  | Escape Character (Backslash) .....                                    | 231 |
| C.2.2.2  | Single-Quotes .....                                                   | 231 |
| C.2.2.3  | Double-Quotes .....                                                   | 231 |
| C.2.3    | Token Recognition .....                                               | 233 |
| C.2.3.1  | Alias Substitution .....                                              | 233 |
| C.2.4    | Reserved Words .....                                                  | 234 |
| C.2.5    | Parameters and Variables .....                                        | 234 |
| C.2.5.1  | Positional Parameters .....                                           | 234 |
| C.2.5.2  | Special Parameters .....                                              | 234 |
| C.2.5.3  | Shell Variables .....                                                 | 235 |
| C.2.6    | Word Expansions .....                                                 | 237 |
| C.2.6.1  | Tilde Expansion .....                                                 | 237 |
| C.2.6.2  | Parameter Expansion .....                                             | 238 |
| C.2.6.3  | Command Substitution .....                                            | 239 |
| C.2.6.4  | Arithmetic Expansion .....                                            | 240 |
| C.2.6.5  | Field Splitting .....                                                 | 241 |
| C.2.6.6  | Pathname Expansion .....                                              | 242 |
| C.2.6.7  | Quote Removal .....                                                   | 242 |
| C.2.7    | Redirection .....                                                     | 242 |
| C.2.7.1  | Redirecting Input .....                                               | 243 |
| C.2.7.2  | Redirecting Output .....                                              | 243 |
| C.2.7.3  | Appending Redirected Output .....                                     | 243 |
| C.2.7.4  | Here-Document .....                                                   | 243 |
| C.2.7.5  | Duplicating an Input File Descriptor .....                            | 243 |
| C.2.7.6  | Duplicating an Output File Descriptor .....                           | 243 |
| C.2.7.7  | Open File Descriptors for Reading and Writing .....                   | 244 |
| C.2.8    | Exit Status and Errors .....                                          | 244 |
| C.2.8.1  | Consequences of Shell Errors .....                                    | 244 |
| C.2.8.2  | Exit Status for Commands .....                                        | 244 |
| C.2.9    | Shell Commands .....                                                  | 244 |
| C.2.9.1  | Simple Commands .....                                                 | 245 |
| C.2.9.2  | Pipelines .....                                                       | 247 |
| C.2.9.3  | Lists .....                                                           | 247 |
| C.2.9.4  | Compound Commands .....                                               | 249 |
| C.2.9.5  | Function Definition Command .....                                     | 250 |
| C.2.10   | Shell Grammar .....                                                   | 251 |
| C.2.10.1 | Shell Grammar Lexical Conventions .....                               | 252 |
| C.2.10.2 | Shell Grammar Rules .....                                             | 252 |

|                 |                                                          |            |
|-----------------|----------------------------------------------------------|------------|
| C.2.11          | Signals and Error Handling .....                         | 253        |
| C.2.12          | Shell Execution Environment .....                        | 253        |
| C.2.13          | Pattern Matching Notation .....                          | 253        |
| C.2.13.1        | Patterns Matching a Single Character .....               | 253        |
| C.2.13.2        | Patterns Matching Multiple Characters .....              | 254        |
| C.2.13.3        | Patterns Used for Filename Expansion .....               | 254        |
| C.2.14          | Special Built-In Utilities .....                         | 255        |
| C.3             | Batch Environment Services and Utilities .....           | 255        |
| C.3.1           | Batch General Concepts .....                             | 258        |
| C.3.2           | Batch Services .....                                     | 260        |
| C.3.3           | Common Behavior for Batch Environment Utilities .....    | 261        |
| C.4             | Utilities .....                                          | 261        |
| <b>Part</b>     | <b>D Portability Considerations .....</b>                | <b>265</b> |
| <b>Appendix</b> | <b>D Portability Considerations (Informative) .....</b>  | <b>267</b> |
| D.1             | User Requirements .....                                  | 267        |
| D.1.1           | Configuration Interrogation .....                        | 268        |
| D.1.2           | Process Management .....                                 | 268        |
| D.1.3           | Access to Data .....                                     | 268        |
| D.1.4           | Access to the Environment .....                          | 268        |
| D.1.5           | Access to Determinism and Performance Enhancements ..... | 268        |
| D.1.6           | Operating System-Dependent Profile .....                 | 268        |
| D.1.7           | I/O Interaction .....                                    | 268        |
| D.1.8           | Internationalization Interaction .....                   | 269        |
| D.1.9           | C-Language Extensions .....                              | 269        |
| D.1.10          | Command Language .....                                   | 269        |
| D.1.11          | Interactive Facilities .....                             | 269        |
| D.1.12          | Accomplish Multiple Tasks Simultaneously .....           | 269        |
| D.1.13          | Complex Data Manipulation .....                          | 269        |
| D.1.14          | File Hierarchy Manipulation .....                        | 269        |
| D.1.15          | Locale Configuration .....                               | 269        |
| D.1.16          | Inter-User Communication .....                           | 270        |
| D.1.17          | System Environment .....                                 | 270        |
| D.1.18          | Printing .....                                           | 270        |
| D.1.19          | Software Development .....                               | 270        |
| D.2             | Portability Capabilities .....                           | 270        |
| D.2.1           | Configuration Interrogation .....                        | 271        |
| D.2.2           | Process Management .....                                 | 271        |
| D.2.3           | Access to Data .....                                     | 272        |
| D.2.4           | Access to the Environment .....                          | 272        |
| D.2.5           | Bounded (Realtime) Response .....                        | 273        |
| D.2.6           | Operating System-Dependent Profile .....                 | 273        |
| D.2.7           | I/O Interaction .....                                    | 273        |
| D.2.8           | Internationalization Interaction .....                   | 273        |
| D.2.9           | C-Language Extensions .....                              | 274        |
| D.2.10          | Command Language .....                                   | 274        |
| D.2.11          | Interactive Facilities .....                             | 274        |

|                        |                                                          |            |
|------------------------|----------------------------------------------------------|------------|
| D.2.12                 | Accomplish Multiple Tasks Simultaneously .....           | 275        |
| D.2.13                 | Complex Data Manipulation.....                           | 275        |
| D.2.14                 | File Hierarchy Manipulation .....                        | 275        |
| D.2.15                 | Locale Configuration.....                                | 276        |
| D.2.16                 | Inter-User Communication.....                            | 276        |
| D.2.17                 | System Environment.....                                  | 276        |
| D.2.18                 | Printing.....                                            | 276        |
| D.2.19                 | Software Development .....                               | 277        |
| D.2.20                 | Future Growth .....                                      | 277        |
| D.3                    | Profiling Considerations.....                            | 277        |
| D.3.1                  | Configuration Options.....                               | 277        |
| D.3.2                  | Configuration Options (Shell and Utilities).....         | 278        |
| D.3.3                  | Configurable Limits.....                                 | 279        |
| D.3.4                  | Configuration Options (System Interfaces) .....          | 280        |
| D.3.5                  | Configurable Limits.....                                 | 285        |
| D.3.6                  | Optional Behavior.....                                   | 288        |
| <b>Part E</b>          | <b>Subprofiling Considerations .....</b>                 | <b>289</b> |
| <b>Appendix E</b>      | <b>Subprofiling Considerations (Informative).....</b>    | <b>291</b> |
| E.1                    | Subprofiling Option Groups.....                          | 291        |
|                        | <b>Index.....</b>                                        | <b>297</b> |
| <b>List of Figures</b> |                                                          |            |
| B-1                    | Example of a System with Typed Memory .....              | 122        |
| B-2                    | Trace System Overview: for Offline Analysis.....         | 184        |
| B-3                    | Trace System Overview: for Online Analysis.....          | 185        |
| B-4                    | Trace System Overview: States of a Trace Stream .....    | 187        |
| B-5                    | Trace Another Process.....                               | 197        |
| B-6                    | Trace Name Space Overview: With Third-Party Library..... | 198        |
| <b>List of Tables</b>  |                                                          |            |
| A-1                    | Historical Practice for Symbolic Links .....             | 30         |

# Foreword

## Structure of the Standard

This standard was originally developed by the Austin Group, a joint working group of members of the IEEE, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1, as one of the four volumes of IEEE Std 1003.1-2001. The standard was approved by ISO and IEC and published in four parts, correlating to the original volumes.

A mapping of the parts to the volumes is shown below:

| <b>ISO/IEC 9945<br/>Part</b> | <b>IEEE Std 1003.1<br/>Volume</b> | <b>Description</b>                                                                                                                                                                                                                           |
|------------------------------|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 9945-1                       | Base Definitions                  | Includes general terms, concepts, and interfaces common to all parts of ISO/IEC 9945, including utility conventions and C-language header definitions.                                                                                       |
| 9945-2                       | System Interfaces                 | Includes definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery.                             |
| 9945-3                       | Shell and Utilities               | Includes definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs.                                                                         |
| 9945-4                       | Rationale                         | Includes extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of ISO/IEC 9945 and why features were included or discarded by the standard developers. |

All four parts comprise the entire standard, and are intended to be used together to accommodate significant internal referencing among them. POSIX-conforming systems are required to support all four parts.

# Introduction

**Note:** This introduction is not part of IEEE Std 1003.1-2001, Standard for Information Technology — Portable Operating System Interface (POSIX).

This standard has been jointly developed by the IEEE and The Open Group. It is simultaneously an IEEE Standard, an ISO/IEC Standard, and an Open Group Technical Standard.

## **The Austin Group**

This standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.<sup>3</sup> The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group has been to revise, combine, and update the following standards: ISO/IEC 9945-1, ISO/IEC 9945-2, IEEE Std 1003.1, IEEE Std 1003.2, and the Base Specifications of The Open Group Single UNIX Specification.

After two initial meetings, an agreement was signed in July 1999 between The Open Group and the Institute of Electrical and Electronics Engineers (IEEE), Inc., to formalize the project with the first draft of the revised specifications being made available at the same time. Under this agreement, The Open Group and IEEE agreed to share joint copyright of the resulting work. The Open Group has provided the chair and secretariat for the Austin Group.

The base document for the revision was The Open Group's Base volumes of its Single UNIX Specification, Version 2. These were selected since they were a superset of the existing POSIX.1 and POSIX.2 specifications and had some organizational aspects that would benefit the audience for the new revision.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group's Technical Standard designation, and an ISO/IEC designation. This set of specifications forms the core of the Single UNIX Specification, Version 3.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see <http://www.opengroup.org/austin>.

---

3. The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.



### Background

The developers of this standard represent a cross section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, this standard describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application writers to write portable applications—it was developed with that goal in mind—it has been designated POSIX,<sup>4</sup> an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of standards.

### Audience

The intended audience for this standard is all persons concerned with an industry-wide standard operating system based on the UNIX system. This includes at least four groups of people:

1. Persons buying hardware and software systems
2. Persons managing companies that are deciding on future corporate computing directions
3. Persons implementing operating systems, and especially
4. Persons developing applications where portability is an objective

### Purpose

Several principles guided the development of this standard:

- Application-Oriented

The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. This standard codifies the common, existing definition of the UNIX system.

- Interface, Not Implementation

This standard defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.

---

4. The name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

- Source, Not Object, Portability

This standard has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. This standard does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.

- The C Language

The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.

- No Superuser, No System Administration

There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in this standard. This standard is also not concerned with hardware constraints or system maintenance.

- Minimal Interface, Minimally Defined

In keeping with the historical design principles of the UNIX system, the mandatory core facilities of this standard have been kept as minimal as possible. Additional capabilities have been added as optional extensions.

- Broadly Implementable

The developers of this standard endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:

1. All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
2. Compatible systems that are not derived from the original UNIX system code
3. Emulations hosted on entirely different operating systems
4. Networked systems
5. Distributed systems
6. Systems running on a broad range of hardware

No direct references to this goal appear in this standard, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations

When the original version of IEEE Std 1003.1 was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and the subsequent revisions and addenda to all of them have consolidated this consensus, and this revision reflects the significantly increased level of consensus arrived at since the original versions. The earlier standards and their modifications specified a number of areas where consensus had not been reached before, and these are now reflected in this revision. The authors of the original versions tried, as much as possible, to follow the principles below

when creating new specifications:

1. By standardizing an interface like one in an historical implementation; for example, directories
2. By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended *tar* format defined in the *pax* utility
3. By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the *sigaction()* function

This revision tries to minimize the number of changes required to implementations which conform to the earlier versions of the approved standards to bring them into conformance with the current standard. Specifically, the scope of this work excluded doing any “new” work, but rather collecting into a single document what had been spread across a number of documents, and presenting it in what had been proven in practice to be a more effective way. Some changes to prior conforming implementations were unavoidable, primarily as a consequence of resolving conflicts found in prior revisions, or which became apparent when bringing the various pieces together.

However, since it references the 1999 version of the ISO C standard, and no longer supports “Common Usage C”, there are a number of unavoidable changes. Applications portability is similarly affected.

This standard is specifically not a codification of a particular vendor’s product.

It should be noted that implementations will have different kinds of extensions. Some will reflect “historical usage” and will be preserved for execution of pre-existing applications. These functions should be considered “obsolescent” and the standard functions used for new applications. Some extensions will represent functions beyond the scope of this standard. These need to be used with careful management to be able to adapt to future extensions of this standard and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code

A goal of this standard was to minimize additional work for the developers of applications. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

### **This Standard**

This standard defines the Portable Operating System Interface (POSIX) requirements and consists of the following volumes:

- Base Definitions
- Shell and Utilities
- System Interfaces
- Rationale (Informative) (this volume)

**This Volume**

This volume is being published to assist in the process of review. It contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers. It also contains notes of interest to application programmers on recommended programming practices, emphasizing the consequences of some aspects of this standard that may not be immediately apparent.

This volume is organized in parallel to the normative volumes of this standard, with a separate part for each of the three normative volumes.

Within this volume, the following terms are used:

**base standard**

The portions of this standard that are not optional, equivalent to the definitions of *classic* POSIX.1 and POSIX.2.

**POSIX.0**

Although this term is not used in the normative text of this standard, it is used in this volume to refer to IEEE Std 1003.0-1995.

**POSIX.1b**

Although this term is not used in the normative text of this standard, it is used in this volume to refer to the elements of the POSIX Realtime Extension amendment. (This was earlier referred to as POSIX.4 during the standard development process.)

**POSIX.1c**

Although this term is not used in the normative text of this standard, it is used in this volume to refer to the POSIX Threads Extension amendment. (This was earlier referred to as POSIX.4a during the standard development process.)

**standard developers**

The individuals and companies in the development organizations responsible for this standard: the IEEE P1003.1 working groups, The Open Group Base working group, advised by the hundreds of individual technical experts who balloted the draft standards within the Austin Group, and the member bodies and technical experts of ISO/IEC JTC 1/SC22/WG15.

**XSI extension**

The portions of this standard addressing the extension added for support of the Single UNIX Specification.

# *Participants*

IEEE Std 1003.1-2001 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/SC22 WG15.

## **The Austin Group**

At the time of approval, the membership of the Austin Group was as follows:

**Andrew Josey**, Chair

**Donald W. Cragun**, Organizational Representative, IEEE PASC

**Nicholas Stoughton**, Organizational Representative, ISO/SC22 WG15

**Mark Brown**, Organizational Representative, The Open Group

**Cathy Hughes**, Technical Editor

## **Austin Group Technical Reviewers**

|                    |                       |                    |
|--------------------|-----------------------|--------------------|
| Peter Anvin        | Michael Gonzalez      | Sandra O'Donnell   |
| Bouazza Bachar     | Joseph M. Gwinn       | Frank Prindle      |
| Theodore P. Baker  | Jon Hitchcock         | Curtis Royster Jr. |
| Walter Briscoe     | Yvette Ho Sang        | Glen Seeds         |
| Mark Brown         | Cathy Hughes          | Keld Jorn Simonsen |
| Dave Butenhof      | Lowell G. Johnson     | Raja Srinivasan    |
| Geoff Clare        | Andrew Josey          | Nicholas Stoughton |
| Donald W. Cragun   | Michael Kavanaugh     | Donn S. Terry      |
| Lee Damico         | David Korn            | Fred Tydeman       |
| Ulrich Drepper     | Marc Aurele La France | Peter Van Der Veen |
| Paul Eggert        | Jim Meyering          | James Youngman     |
| Joanna Farley      | Gary Miller           | Jim Zepeda         |
| Clive D.W. Feather | Finnbarr P. Murphy    | Jason Zions        |
| Andrew Gollan      | Joseph S. Myers       |                    |

# **Austin Group Working Group Members**

|                            |                       |                      |
|----------------------------|-----------------------|----------------------|
| Harold C. Adams            | Michael Gonzalez      | Sandra O'Donnell     |
| Peter Anvin                | Karen D. Gordon       | Frank Prindle        |
| Pierre-Jean Arcos          | Joseph M. Gwinn       | Francois Riche       |
| Jay Ashford                | Steven A. Haaser      | John D. Riley        |
| Bouazza Bachar             | Charles E. Hammons    | Andrew K. Roach      |
| Theodore P. Baker          | Chris J. Harding      | Helmut Roth          |
| Robert Barned              | Barry Hedquist        | Jaideep Roy          |
| Joel Berman                | Vincent E. Henley     | Curtis Royster Jr.   |
| David J. Blackwood         | Karl Heubaum          | Stephen C. Schwarm   |
| Shirley Bockstahler-Brandt | Jon Hitchcock         | Glen Seeds           |
| James Bottomley            | Yvette Ho Sang        | Richard Seibel       |
| Walter Briscoe             | Niklas Holsti         | David L. Shroads Jr. |
| Andries Brouwer            | Thomas Hosmer         | W. Olin Sibert       |
| Mark Brown                 | Cathy Hughes          | Keld Jorn Simonsen   |
| Eric W. Burger             | Jim D. Isaak          | Curtis Smith         |
| Alan Burns                 | Lowell G. Johnson     | Raja Srinivasan      |
| Andries Brouwer            | Michael B. Jones      | Nicholas Stoughton   |
| Dave Butenhof              | Andrew Josey          | Marc J. Teller       |
| Keith Chow                 | Michael J. Karels     | Donn S. Terry        |
| Geoff Clare                | Michael Kavanaugh     | Fred Tydeman         |
| Donald W. Cragun           | David Korn            | Mark-Rene Uchida     |
| Lee Damico                 | Steven Kramer         | Scott A. Valcourt    |
| Juan Antonio De La Puente  | Thomas M. Kurihara    | Peter Van Der Veen   |
| Ming De Zhou               | Marc Aurele La France | Michael W. Vannier   |
| Steven J. Dovich           | C. Douglass Locke     | Eric Vought          |
| Richard P. Draves          | Nick Maclaren         | Frederick N. Webb    |
| Ulrich Drepper             | Roger J. Martin       | Paul A.T. Wolfgang   |
| Paul Eggert                | Craig H. Meyer        | Garrett A. Wollman   |
| Philip H. Enslow           | Jim Meyering          | James Youngman       |
| Joanna Farley              | Gary Miller           | Oren Yuen            |
| Clive D.W. Feather         | Finnbarr P. Murphy    | Janusz Zalewski      |
| Pete Forman                | Joseph S. Myers       | Jim Zepeda           |
| Mark Funkenhauser          | John Napier           | Jason Zions          |
| Lois Goldthwaite           | Peter E. Obermayer    |                      |
| Andrew Gollan              | James T. Oblinger     |                      |

**The Open Group**

When The Open Group approved the Base Specifications, Issue 6 on 12 September 2001, the membership of The Open Group Base Working Group was as follows:

**Andrew Josey**, Chair

**Finnbarr P. Murphy**, Vice-Chair

**Mark Brown**, Austin Group Liaison

**Cathy Hughes**, Technical Editor

**Base Working Group Members**

Bouazza Bachar

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Joanna Farley

Andrew Gollan

Karen D. Gordon

Gary Miller

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

## IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, the membership of the committees was as follows:

### Portable Applications Standards Committee (PASC)

**Lowell G. Johnson**, Chair  
**Joseph M. Gwinn**, Vice-Chair  
**Jay Ashford**, Functional Chair  
**Andrew Josey**, Functional Chair  
**Curtis Royster Jr.**, Functional Chair  
**Nicholas Stoughton**, Secretary

### Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001. Balloters may have voted for approval, disapproval, or abstention:

|                            |                    |                      |
|----------------------------|--------------------|----------------------|
| Harold C. Adams            | Steven A. Haaser   | Frank Prindle        |
| Pierre-Jean Arcos          | Charles E. Hammons | Francois Riche       |
| Jay Ashford                | Chris J. Harding   | John D. Riley        |
| Theodore P. Baker          | Barry Hedquist     | Andrew K. Roach      |
| Robert Barned              | Vincent E. Henley  | Helmut Roth          |
| David J. Blackwood         | Karl Heubaum       | Jaideep Roy          |
| Shirley Bockstahler-Brandt | Niklas Holsti      | Curtis Royster Jr.   |
| James Bottomley            | Thomas Hosmer      | Stephen C. Schwarm   |
| Mark Brown                 | Jim D. Isaak       | Richard Seibel       |
| Eric W. Burger             | Lowell G. Johnson  | David L. Shroads Jr. |
| Alan Burns                 | Michael B. Jones   | W. Olin Sibert       |
| Dave Butenhof              | Andrew Josey       | Keld Jorn Simonsen   |
| Keith Chow                 | Michael J. Karels  | Nicholas Stoughton   |
| Donald W. Cragun           | Steven Kramer      | Donn S. Terry        |
| Juan Antonio De La Puente  | Thomas M. Kurihara | Mark-Rene Uchida     |
| Ming De Zhou               | C. Douglass Locke  | Scott A. Valcourt    |
| Steven J. Dovich           | Roger J. Martin    | Michael W. Vannier   |
| Richard P. Draves          | Craig H. Meyer     | Frederick N. Webb    |
| Philip H. Enslow           | Finnbarr P. Murphy | Paul A.T. Wolfgang   |
| Michael Gonzalez           | John Napier        | Oren Yuen            |
| Karen D. Gordon            | Peter E. Obermayer | Janusz Zalewski      |
| Joseph M. Gwinn            | James T. Oblinger  |                      |

The following organizational representative voted on this standard:

**Andrew Josey**, X/Open Company Ltd.



### IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, it had the following membership:

**Donald N. Heirman**, Chair

**James T. Carlo**, Vice-Chair

**Judith Gorman**, Secretary

Satish K. Aggarwal

Mark D. Bowman

Gary R. Engmann

Harold E. Epstein

H. Landis Floyd

Jay Forster\*

Howard M. Frazier

Ruben D. Garzon

James H. Gurney

Richard J. Holleman

Lowell G. Johnson

Robert J. Kennelly

Joseph L. Koepfinger\*

Peter H. Lips

L. Bruce McClung

Daleep C. Mohla

James W. Moore

Robert F. Munzner

Ronald C. Petersen

Gerald H. Peterson

John B. Posey

Gary S. Robinson

Akio Tojo

Donald W. Zipse

Also included are the following non-voting IEEE-SA Standards Board liaisons:

**Alan Cookson**, NIST Representative

**Donald R. Volzka**, TAB Representative

**Yvette Ho Sang**, **Don Messina**, **Savoula Amanatidis**, IEEE Project Editors

---

\* Member Emeritus

IEEE Std 1003.1-2001/Cor 1-2002 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/IEC JTC 1/SC22/WG15.

### **The Austin Group**

At the time of approval, the membership of the Austin Group was as follows:

**Andrew Josey**, Chair

**Donald W. Cragun**, Organizational Representative, IEEE PASC

**Nicholas Stoughton**, Organizational Representative, ISO/IEC JTC 1/SC22/WG15

**Mark Brown**, Organizational Representative, The Open Group

**Cathy Fox**, Technical Editor

### **Austin Group Technical Reviewers**

Theodore P. Baker

Julian Blake

Andries Brouwer

Mark Brown

Dave Butenhof

Geoff Clare

Donald W. Cragun

Ken Dawson

Ulrich Drepper

Larry Dwyer

Paul Eggert

Joanna Farley

Clive D.W. Feather

Cathy Fox

Mark Funkenhauser

Lois Goldthwaite

Andrew Gollan

Michael Gonzalez

Bruno Haible

Ben Harris

Jon Hitchcock

Andreas Jaeger

Andrew Josey

Jonathan Lennox

Nick Maclaren

Jack McCann

Wilhelm Mueller

Joseph S. Myers

Frank Prindle

Kenneth Raeburn

Tim Robbins

Glen Seeds

Matthew Seitz

Keld Jorn Simonsen

Nicholas Stoughton

Alexander Terekhov

Donn S. Terry

Mike Wilson

Garrett A. Wollman

Mark Ziegast

**Austin Group Working Group Members**

Harold C. Adams  
Alejandro Alonso  
Jay Ashford  
Theodore P. Baker  
David J. Blackwood  
Julian Blake  
Mitchell Bonnett  
Andries Brouwer  
Mark Brown  
Eric W. Burger  
Alan Burns  
Dave Butenhof  
Keith Chow  
Geoff Clare  
Luis Cordova  
Donald W. Cragun  
Dragan Cvetkovic  
Lee Damico  
Ken Dawson  
Jeroen Dekkers  
Juan Antonio De La Puente  
Steven J. Dovich  
Ulrich Drepper  
Dr. Sourav Dutta  
Larry Dwyer  
Paul Eggert  
Joanna Farley

Clive D.W. Feather  
Yaacov Fenster  
Cathy Fox  
Mark Funkenhauser  
Lois Goldthwaite  
Andrew Gollan  
Michael Gonzalez  
Karen D. Gordon  
Scott Gudgel  
Joseph M. Gwinn  
Steven A. Haaser  
Bruno Haible  
Charles E. Hammons  
Bryan Harold  
Ben Harris  
Barry Hedquist  
Karl Heubaum  
Jon Hitchcock  
Andreas Jaeger  
Andrew Josey  
Kenneth Lang  
Pi-Cheng Law  
Jonathan Lennox  
Nick Maclaren  
Roger J. Martin  
Jack McCann  
George Miao

Wilhelm Mueller  
Finnbarr P. Murphy  
Joseph S. Myers  
Alexey Neyman  
Charles Ngethe  
Peter Petrov  
Frank Prindle  
Vikram Punj  
Kenneth Raeburn  
Francois Riche  
Tim Robbins  
Curtis Royster Jr.  
Diane Schleicher  
Gil Shultz  
Stephen C. Schwarm  
Glen Seeds  
Matthew Seitz  
Keld Jorn Simonsen  
Doug Stevenson  
Nicholas Stoughton  
Alexander Terekhov  
Donn S. Terry  
Mike Wilson  
Garrett A. Wollman  
Oren Yuen  
Mark Ziegast

### **The Open Group**

When The Open Group approved the Base Specifications, Issue 6, Technical Corrigendum 1 on 7 February 2003, the membership of The Open Group Base Working Group was as follows:

**Andrew Josey**, Chair

**Finnbarr P. Murphy**, Vice-Chair

**Mark Brown**, Austin Group Liaison

**Cathy Fox**, Technical Editor

### **Base Working Group Members**

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Ulrich Drepper

Joanna Farley

Andrew Gollan

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

### IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001/Cor 1-2002 on 11 December 2002, the membership of the committees was as follows:

#### Portable Applications Standards Committee (PASC)

**Lowell G. Johnson**, Chair

**Joseph M. Gwinn**, Vice-Chair

**Jay Ashford**, Functional Chair

**Andrew Josey**, Functional Chair

**Curtis Royster Jr.**, Functional Chair

**Nicholas Stoughton**, Secretary

#### Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001/Cor 1-2002. Balloters may have voted for approval, disapproval, or abstention:

Alejandro Alonso  
Jay Ashford  
David J. Blackwood  
Julian Blake  
Mitchell Bonnett  
Mark Brown  
Dave Butenhof  
Keith Chow  
Luis Cordova  
Donald W. Cragun  
Steven J. Dovich  
Dr. Sourav Dutta  
Yaacov Fenster

Michael Gonzalez  
Scott Gudgel  
Charles E. Hammons  
Bryan Harold  
Barry Hedquist  
Karl Heubaum  
Lowell G. Johnson  
Andrew Josey  
Kenneth Lang  
Pi-Cheng Law  
George Miao  
Roger J. Martin  
Finnbarr P. Murphy

Charles Ngethe  
Peter Petrov  
Frank Prindle  
Vikram Punj  
Francois Riche  
Curtis Royster Jr.  
Diane Schleicher  
Stephen C. Schwarm  
Gil Shultz  
Nicholas Stoughton  
Donn S. Terry  
Oren Yuen  
Juan A. de la Puente

## IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 / Cor 1-2002 on 11 December 2002, the membership was as follows:

**James T. Carlo**, Chair

**James H. Gurney**, Vice-Chair

**Judith Gorman**, Secretary

Sid Bennett

H. Stephen Berger

Clyde R. Camp

Richard DeBlasio

Harold E. Epstein

Julian Forster\*

Howard M. Frazier

Toshio Fukuda

Arnold M. Greenspan

Raymond Hapeman

Donald M. Heirman

Richard H. Hulett

Lowell G. Johnson

Joseph L. Koepfinger\*

Peter H. Lips

Nader Mehravari

Daleep C. Mohla

William J. Moylan

Malcolm V. Thaden

Geoffrey O. Thompson

Howard L. Wolfman

Don Wright

Also included are the following non-voting IEEE-SA Standards Board liaisons:

**Alan Cookson**, NIST Representative

**Satish K. Aggarwal**, NRC Representative

**Savoula Amanatidis**, IEEE Standards Managing Editor

---

\* Member Emeritus

# *Trademarks*

The following information is given for the convenience of users of this standard and does not constitute endorsement of these products by The Open Group or the IEEE. There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

1003.1<sup>™</sup> is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

AIX<sup>®</sup> is a registered trademark of IBM Corporation.

AT&T<sup>®</sup> is a registered trademark of AT&T in the U.S.A. and other countries.

BSD<sup>™</sup> is a trademark of the University of California, Berkeley, U.S.A.

Hewlett-Packard<sup>®</sup>, HP<sup>®</sup>, and HP-UX<sup>®</sup> are registered trademarks of Hewlett-Packard Company.

IBM<sup>®</sup> is a registered trademark of International Business Machines Corporation.

The Open Group and Boundaryless Information Flow are trademarks and UNIX is a registered trademark of The Open Group in the United States and other countries. All other trademarks are the property of their respective owners.

POSIX<sup>®</sup> is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Sun<sup>®</sup> and Sun Microsystems<sup>®</sup> are registered trademarks of Sun Microsystems, Inc.

/usr/group<sup>®</sup> is a registered trademark of UniForum, the International Network of UNIX System Users.

# *Acknowledgements*

The contributions of the following organizations to the development of IEEE Std 1003.1-2001 are gratefully acknowledged:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.
- The SC22 WG14 Committees.

This standard was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO SC22 WG15.



# *Referenced Documents*

## **Normative References**

Normative references for this standard are defined in the Base Definitions volume.

## **Informative References**

The following documents are referenced in this standard:

1984 /usr/group Standard

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

Almasi and Gottlieb

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

ANSI C

American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.226-1994

American National Standard for Information Systems: Standard X3.226-1994, Programming Language Common LISP.

Brawer

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989, ISBN: 0-12-128470-0.

DeRemer and Pennello Article

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets*, SigPlan Notices, Volume 15, No. 8, August 1979.

Draft ANSI X3J11.1

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-1

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

FIPS 151-2

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

HP-UX Manual

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

IEC 60559: 1989

IEC 60559: 1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously designated IEC 559: 1989).

IEEE Std 754-1985

IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic.

IEEE Std 854-1987

IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.

- IEEE Std 1003.9-1992  
IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.
- IETF RFC 791  
Internet Protocol, Version 4 (IPv4), September 1981.
- IETF RFC 819  
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982.
- IETF RFC 822  
Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982.
- IETF RFC 919  
Broadcasting Internet Datagrams, J. Mogul, October 1984.
- IETF RFC 920  
Domain Requirements, J. Postel, J. Reynolds, October 1984.
- IETF RFC 921  
Domain Name System Implementation Schedule, J. Postel, October 1984.
- IETF RFC 922  
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984.
- IETF RFC 1034  
Domain Names — Concepts and Facilities, P. Mockapetris, November 1987.
- IETF RFC 1035  
Domain Names — Implementation and Specification, P. Mockapetris, November 1987.
- IETF RFC 1123  
Requirements for Internet Hosts — Application and Support, R. Braden, October 1989.
- IETF RFC 1886  
DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson, December 1995.
- IETF RFC 2045  
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996.
- IETF RFC 2181  
Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997.
- IETF RFC 2373  
Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998.
- IETF RFC 2460  
Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998.
- Internationalisation Guide  
Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.
- ISO C (1990)  
ISO/IEC 9899:1990, Programming Languages — C, including Amendment 1:1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).

### ISO 2375:1985

ISO 2375:1985, Data Processing — Procedure for Registration of Escape Sequences.

### ISO 8652:1987

ISO 8652:1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).

### ISO/IEC 1539:1990

ISO/IEC 1539:1990, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

### ISO/IEC 4873:1991

ISO/IEC 4873:1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.

### ISO/IEC 6429:1992

ISO/IEC 6429:1992, Information Technology — Control Functions for Coded Character Sets.

### ISO/IEC 6937:1994

ISO/IEC 6937:1994, Information Technology — Coded Character Set for Text Communication — Latin Alphabet.

### ISO/IEC 8802-3:1996

ISO/IEC 8802-3:1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

### ISO/IEC 8859

ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

Part 1: Latin Alphabet No. 1

Part 2: Latin Alphabet No. 2

Part 3: Latin Alphabet No. 3

Part 4: Latin Alphabet No. 4

Part 5: Latin/Cyrillic Alphabet

Part 6: Latin/Arabic Alphabet

Part 7: Latin/Greek Alphabet

Part 8: Latin/Hebrew Alphabet

Part 9: Latin Alphabet No. 5

Part 10: Latin Alphabet No. 6

Part 13: Latin Alphabet No. 7

Part 14: Latin Alphabet No. 8

Part 15: Latin Alphabet No. 9

### ISO POSIX-1:1996

ISO/IEC 9945-1:1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Std 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.

### ISO POSIX-2:1993

ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2-1992, as amended by ANSI/IEEE Std 1003.2a-1992).

#### Issue 1

X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

#### Issue 2

X/Open Portability Guide, January 1987:

- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
- Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)

#### Issue 3

X/Open Specification, 1988, 1989, February 1992:

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

#### Issue 4

CAE Specification, July 1992, published by The Open Group:

- System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
- Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
- System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

#### Issue 4, Version 2

CAE Specification, August 1994, published by The Open Group:

- System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

#### Issue 5

Technical Standard, February 1997, published by The Open Group:

- System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

#### Knuth Article

Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control, Volume 8, No. 6, October 1965.

### KornShell

Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming Language*, March 1995, Prentice Hall.

### MSE Working Draft

Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

### POSIX.0: 1995

IEEE Std 1003.0-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical to ISO/IEC TR 14252).

### POSIX.1: 1988

IEEE Std 1003.1-1988, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

### POSIX.1: 1990

IEEE Std 1003.1-1990, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

### POSIX.1a

P1003.1a, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — (C Language) Amendment.

### POSIX.1d: 1999

IEEE Std 1003.1d-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 4: Additional Realtime Extensions [C Language].

### POSIX.1g: 2000

IEEE Std 1003.1g-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 6: Protocol-Independent Interfaces (PII).

### POSIX.1j: 2000

IEEE Std 1003.1j-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 5: Advanced Realtime Extensions [C Language].

### POSIX.1q: 2000

IEEE Std 1003.1q-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 7: Tracing [C Language].

### POSIX.2b

P1003.2b, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment.

### POSIX.2d:-1994

IEEE Std 1003.2d-1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.

POSIX.13:-1998

IEEE Std 1003.13:1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.

Sarwate Article

Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications of the ACM, Volume 30, No. 8, August 1988.

Sprunt, Sha, and Lehoczky

Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*, The Journal of Real-Time Systems, Volume 1, 1989, Pages 27-60.

SVID, Issue 1

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.

SVID, Issue 2

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.

SVID, Issue 3

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.

The AWK Programming Language

Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.

UNIX Programmer's Manual

American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.

XNS, Issue 4

CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.

XNS, Issue 5

CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XNS, Issue 5.2

Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.

X/Open Curses, Issue 4, Version 2

CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.

Yacc

*Yacc: Yet Another Compiler Compiler*, Stephen C. Johnson, 1978.

**Source Documents**

Parts of the following documents were used to create the base documents for this standard:

**AIX 3.2 Manual**

AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).

**OSF/1**

OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).

**OSF AES**

Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

**System V Release 2.0**

— UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).

— UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).

**System V Release 4.2**

Operating System API Reference, UNIX SVR4.2 (1992) (ISBN: 0-13-017658-3).





# *Rationale (Informative)*

## **Part A:**

### **Base Definitions**

*The Open Group*

*The Institute of Electrical and Electronics Engineers, Inc.*



# Rationale for Base Definitions

## A.1 Introduction

### A.1.1 Scope

IEEE Std 1003.1-2001 is one of a family of standards known as POSIX. The family of standards extends to many topics; IEEE Std 1003.1-2001 is known as POSIX.1 and consists of both operating system interfaces and shell and utilities. IEEE Std 1003.1-2001 is technically identical to The Open Group Base Specifications, Issue 6, which comprise the core volumes of the Single UNIX Specification, Version 3.

#### Scope of IEEE Std 1003.1-2001

The (paraphrased) goals of this development were to produce a single common revision to the overlapping POSIX.1 and POSIX.2 standards, and the Single UNIX Specification, Version 2. As such, the scope of the revision includes the scopes of the original documents merged.

Since the revision includes merging the Base volumes of the Single UNIX Specification, many features that were previously not “adopted” into earlier revisions of POSIX.1 and POSIX.2 are now included in IEEE Std 1003.1-2001. In most cases, these additions are part of the XSI extension; in other cases the standard developers decided that now was the time to migrate these to the base standard.

The Single UNIX Specification programming environment provides a broad-based functional set of interfaces to support the porting of existing UNIX applications and the development of new applications. The environment also supports a rich set of tools for application development.

The majority of the obsolescent material from the existing POSIX.1 and POSIX.2 standards, and material marked LEGACY from The Open Group’s Base specifications, has been removed in this revision. New members of the Legacy Option Group have been added, reflecting the advance in understanding of what is required.

The following IEEE standards have been added to the base documents in this revision:

- IEEE Std 1003.1d-1999
- IEEE Std 1003.1j-2000
- IEEE Std 1003.1q-2000
- IEEE P1003.1a draft standard
- IEEE Std 1003.2d-1994
- IEEE P1003.2b draft standard
- Selected parts of IEEE Std 1003.1g-2000

Only selected parts of IEEE Std 1003.1g-2000 were included. This was because there is much duplication between the XNS, Issue 5.2 specification (another base document) and the material from IEEE Std 1003.1g-2000, the former document being aligned with the latest networking specifications for IPv6. Only the following sections of IEEE Std 1003.1g-2000 were considered for inclusion:

- General terms related to sockets (Section 2.2.2)
- Socket concepts (Sections 5.1 through 5.3 inclusive)
- The `pselect()` function (Sections 6.2.2.1 and 6.2.3)
- The `<sys/select.h>` header (Section 6.2)

The following were requirements on IEEE Std 1003.1-2001:

- Backward-compatibility

It was agreed that there should be no breakage of functionality in the existing base documents. This requirement was tempered by changes introduced due to interpretations and corrigenda on the base documents, and any changes introduced in the ISO/IEC 9899:1999 standard (C Language).

- Architecture and n-bit neutral

The common standard should not make any implicit assumptions about the system architecture or size of data types; for example, previously some 32-bit implicit assumptions had crept into the standards.

- Extensibility

It should be possible to extend the common standard without breaking backwards-compatibility. For example, the name space should be reserved and structured to avoid duplication of names between the standard and extensions to it.

## POSIX.1 and the ISO C Standard

Previous revisions of POSIX.1 built upon the ISO C standard by reference only. This revision takes a different approach.

The standard developers believed it essential for a programmer to have a single complete reference place, but recognized that deference to the formal standard had to be addressed for the duplicate interface definitions between the ISO C standard and the Single UNIX Specification.

It was agreed that where an interface has a version in the ISO C standard, the DESCRIPTION section should describe the relationship to the ISO C standard and markings should be added as appropriate to show where the ISO C standard has been extended in the text.

A block of text was added to the start of each affected reference page stating whether the page is aligned with the ISO C standard or extended. Each page was parsed for additions beyond the ISO C standard (that is, including both POSIX and UNIX extensions), and these extensions are marked as CX extensions (for C Extensions).

## FIPS Requirements

The Federal Information Processing Standards (FIPS) are a series of U.S. government procurement standards managed and maintained on behalf of the U.S. Department of Commerce by the National Institute of Standards and Technology (NIST).

The following restrictions have been made in this version of IEEE Std 1003.1 in order to align with FIPS 151-2 requirements:

- The implementation supports `_POSIX_CHOWN_RESTRICTED`.
- The limit `{NGROUPS_MAX}` is now greater than or equal to 8.
- The implementation supports the setting of the group ID of a file (when it is created) to that of the parent directory.

- The implementation supports `_POSIX_SAVED_IDS`.
- The implementation supports `_POSIX_VDISABLE`.
- The implementation supports `_POSIX_JOB_CONTROL`.
- The implementation supports `_POSIX_NO_TRUNC`.
- The `read()` function returns the number of bytes read when interrupted by a signal and does not return `-1`.
- The `write()` function returns the number of bytes written when interrupted by a signal and does not return `-1`.
- In the environment for the login shell, the environment variables `LOGNAME` and `HOME` are defined and have the properties described in IEEE Std 1003.1-2001.
- The value of `{CHILD_MAX}` is now greater than or equal to 25.
- The value of `{OPEN_MAX}` is now greater than or equal to 20.
- The implementation supports the functionality associated with the symbols `CS7`, `CS8`, `CSTOPB`, `PARODD`, and `PARENB` defined in `<termios.h>`.

### A.1.2 Conformance

See Section A.2 (on page 9).

### A.1.3 Normative References

There is no additional rationale provided for this section.

### A.1.4 Terminology

The meanings specified in IEEE Std 1003.1-2001 for the words *shall*, *should*, and *may* are mandated by ISO/IEC directives.

In the Rationale (Informative) volume of IEEE Std 1003.1-2001, the words *shall*, *should*, and *may* are sometimes used to illustrate similar usages in IEEE Std 1003.1-2001. However, the rationale itself does not specify anything regarding implementations or applications.

#### conformance document

As a practical matter, the conformance document is effectively part of the system documentation. Conformance documents are distinguished by IEEE Std 1003.1-2001 so that they can be referred to distinctly.

#### implementation-defined

This definition is analogous to that of the ISO C standard and, together with “undefined” and “unspecified”, provides a range of specification of freedom allowed to the interface implementor.

#### may

The use of *may* has been limited as much as possible, due both to confusion stemming from its ordinary English meaning and to objections regarding the desirability of having as few options as possible and those as clearly specified as possible.

The usage of *can* and *may* were selected to contrast optional application behavior (can) against optional implementation behavior (may).

**shall**

Declarative sentences are sometimes used in IEEE Std 1003.1-2001 as if they included the word *shall*, and facilities thus specified are no less required. For example, the two statements:

1. The *foo()* function shall return zero.
2. The *foo()* function returns zero.

are meant to be exactly equivalent.

**should**

In IEEE Std 1003.1-2001, the word *should* does not usually apply to the implementation, but rather to the application. Thus, the important words regarding implementations are *shall*, which indicates requirements, and *may*, which indicates options.

**obsolescent**

The term “obsolescent” means “do not use this feature in new applications”. The obsolescence concept is not an ideal solution, but was used as a method of increasing consensus: many more objections would be heard from the user community if some of these historical features were suddenly withdrawn without the grace period obsolescence implies. The phrase “may be considered for withdrawal in future revisions” implies that the result of that consideration might in fact keep those features indefinitely if the predominance of applications do not migrate away from them quickly.

**legacy**

The term “legacy” was added for compatibility with the Single UNIX Specification. It means “this feature is historic and optional; do not use this feature in new applications. There are alternative interfaces that are more suitable.”. It is used exclusively for XSI extensions, and includes facilities that were mandatory in previous versions of the base document but are optional in this revision. This is a way to “sunset” the usage of certain functions. Application writers should not rely on the existence of these facilities in new applications, but should follow the migration path detailed in the APPLICATION USAGE sections of the relevant pages.

The terms “legacy” and “obsolescent” are different: a feature marked LEGACY is not recommended for new work and need not be present on an implementation (if the XSI Legacy Option Group is not supported). A feature noted as obsolescent is supported by all implementations, but may be removed in a future revision; new applications should not use these features.

**system documentation**

The system documentation should normally describe the whole of the implementation, including any extensions provided by the implementation. Such documents normally contain information at least as detailed as the specifications in IEEE Std 1003.1-2001. Few requirements are made on the system documentation, but the term is needed to avoid a dangling pointer where the conformance document is permitted to point to the system documentation.

**undefined**

See *implementation-defined*.

**unspecified**

See *implementation-defined*.

The definitions for “unspecified” and “undefined” appear nearly identical at first examination, but are not. The term “unspecified” means that a conforming application may deal with the unspecified behavior, and it should not care what the outcome is. The term

“undefined” says that a conforming application should not do it because no definition is provided for what it does (and implicitly it would care what the outcome was if it tried it). It is important to remember, however, that if the syntax permits the statement at all, it must have some outcome in a real implementation.

Thus, the terms “undefined” and “unspecified” apply to the way the application should think about the feature. In terms of the implementation, it is always “defined”—there is always some result, even if it is an error. The implementation is free to choose the behavior it prefers.

This also implies that an implementation, or another standard, could specify or define the result in a useful fashion. The terms apply to IEEE Std 1003.1-2001 specifically.

The term “implementation-defined” implies requirements for documentation that are not required for “undefined” (or “unspecified”). Where there is no need for a conforming program to know the definition, the term “undefined” is used, even though “implementation-defined” could also have been used in this context. There could be a fourth term, specifying “this standard does not say what this does; it is acceptable to define it in an implementation, but it does not need to be documented”, and undefined would then be used very rarely for the few things for which any definition is not useful. In particular, implementation-defined is used where it is believed that certain classes of application will need to know such details to determine whether the application can be successfully ported to the implementation. Such applications are not always strictly portable, but nevertheless are common and useful; often the requirements met by the application cannot be met without dealing with the issues implied by “implementation-defined”.

In many places IEEE Std 1003.1-2001 is silent about the behavior of some possible construct. For example, a variable may be defined for a specified range of values and behaviors are described for those values; nothing is said about what happens if the variable has any other value. That kind of silence can imply an error in the standard, but it may also imply that the standard was intentionally silent and that any behavior is permitted. There is a natural tendency to infer that if the standard is silent, a behavior is prohibited. That is not the intent. Silence is intended to be equivalent to the term “unspecified”.

The term “application” is not defined in IEEE Std 1003.1-2001; it is assumed to be a part of general computer science terminology.

Three terms used within IEEE Std 1003.1-2001 overlap in meaning: “macro”, “symbolic name”, and “symbolic constant”.

#### **macro**

This usually describes a C preprocessor symbol, the result of the **#define** operator, with or without an argument. It may also be used to describe similar mechanisms in editors and text processors.

#### **symbolic name**

This can also refer to a C preprocessor symbol (without arguments), but is also used to refer to the names for characters in character sets. In addition, it is sometimes used to refer to host names and even filenames.

#### **symbolic constant**

This also refers to a C preprocessor symbol (also without arguments).

In most cases, the difference in semantic content is negligible to nonexistent. Readers should not attempt to read any meaning into the various usages of these terms.

### A.1.5 Portability

To aid the identification of options within IEEE Std 1003.1-2001, a notation consisting of margin codes and shading is used. This is based on the notation used in previous revisions of The Open Group's Base specifications.

The benefit of this approach is a reduction in the number of *if* statements within the running text, that makes the text easier to read, and also an identification to the programmer that they need to ensure that their target platforms support the underlying options. For example, if functionality is marked with THR in the margin, it will be available on all systems supporting the Threads option, but may not be available on some others.

#### A.1.5.1 Codes

This section includes codes for options defined in the Base Definitions volume of IEEE Std 1003.1-2001, Section 2.1.6, Options, and the following additional codes for other purposes:

**CX** This margin code is used to denote extensions beyond the ISO C standard. For interfaces that are duplicated between IEEE Std 1003.1-2001 and the ISO C standard, a CX introduction block describes the nature of the duplication, with any extensions appropriately CX marked and shaded.

Where an interface is added to an ISO C standard header, within the header the interface has an appropriate margin marker and shading (for example, CX, XSI, TSF, and so on) and the same marking appears on the reference page in the SYNOPSIS section. This enables a programmer to easily identify that the interface is extending an ISO C standard header.

**MX** This margin code is used to denote IEC 60559:1989 standard floating-point extensions.

**OB** This margin code is used to denote obsolescent behavior and thus flag a possible future applications portability warning.

**OH** The Single UNIX Specification has historically tried to reduce the number of headers an application has had to include when using a particular interface. Sometimes this was fewer than the base standard, and hence a notation is used to flag which headers are optional if you are using a system supporting the XSI extension.

**XSI** This code is used to denote interfaces and facilities within interfaces only required on systems supporting the XSI extension. This is introduced to support the Single UNIX Specification.

**XSR** This code is used to denote interfaces and facilities within interfaces only required on systems supporting STREAMS. This is introduced to support the Single UNIX Specification, although it is defined in a way so that it can stand alone from the XSI notation.

#### A.1.5.2 Margin Code Notation

Since some features may depend on one or more options, or require more than one option, a notation is used. Where a feature requires support of a single option, a single margin code will occur in the margin. If it depends on two options and both are required, then the codes will appear with a <space> separator. If either of two options are required, then a logical OR is denoted using the ' | ' symbol. If more than two codes are used, a special notation is used.



## A.2 Conformance

The terms “profile” and “profiling” are used throughout this section.

A profile of a standard or standards is a codified set of option selections, such that by being conformant to a profile, particular classes of users are specifically supported.

These conformance definitions are descended from those in the ISO POSIX-1: 1996 standard, but with changes for the following:

- The addition of profiling options, allowing larger profiles of options such as the XSI extension used by the Single UNIX Specification. In effect, it has profiled itself (that is, created a self-profile).
- The addition of a definition of subprofiling considerations, to allow smaller profiles of options.
- The addition of a hierarchy of super-options for XSI; these were formerly known as “Feature Groups” in the System Interfaces and Headers, Issue 5 specification.
- Options from the ISO POSIX-2: 1993 standard are also now included, as IEEE Std 1003.1-2001 merges the functionality from it.

### A.2.1 Implementation Conformance

These definitions allow application developers to know what to depend on in an implementation.

There is no definition of a “strictly conforming implementation”; that would be an implementation that provides *only* those facilities specified by POSIX.1 with no extensions whatsoever. This is because no actual operating system implementation can exist without system administration and initialization facilities that are beyond the scope of POSIX.1.

#### A.2.1.1 Requirements

The word “support” is used in certain instances, rather than “provide”, in order to allow an implementation that has no resident software development facilities, but that supports the execution of a *Strictly Conforming POSIX.1 Application*, to be a *conforming implementation*.

#### A.2.1.2 Documentation

The conformance documentation is required to use the same numbering scheme as POSIX.1 for purposes of cross-referencing. All options that an implementation chooses are reflected in **<limits.h>** and **<unistd.h>**.

Note that the use of “may” in terms of where conformance documents record where implementations may vary, implies that it is not required to describe those features identified as undefined or unspecified.

Other aspects of systems must be evaluated by purchasers for suitability. Many systems incorporate buffering facilities, maintaining updated data in volatile storage and transferring such updates to non-volatile storage asynchronously. Various exception conditions, such as a power failure or a system crash, can cause this data to be lost. The data may be associated with a file that is still open, with one that has been closed, with a directory, or with any other internal system data structures associated with permanent storage. This data can be lost, in whole or part, so that only careful inspection of file contents could determine that an update did not occur.

Also, interrelated file activities, where multiple files and/or directories are updated, or where space is allocated or released in the file system structures, can leave inconsistencies in the relationship between data in the various files and directories, or in the file system itself. Such inconsistencies can break applications that expect updates to occur in a specific sequence, so that updates in one place correspond with related updates in another place.

For example, if a user creates a file, places information in the file, and then records this action in another file, a system or power failure at this point followed by restart may result in a state in which the record of the action is permanently recorded, but the file created (or some of its information) has been lost. The consequences of this to the user may be undesirable. For a user on such a system, the only safe action may be to require the system administrator to have a policy that requires, after any system or power failure, that the entire file system must be restored from the most recent backup copy (causing all intervening work to be lost).

The characteristics of each implementation will vary in this respect and may or may not meet the requirements of a given application or user. Enforcement of such requirements is beyond the scope of POSIX.1. It is up to the purchaser to determine what facilities are provided in an implementation that affect the exposure to possible data or sequence loss, and also what underlying implementation techniques and/or facilities are provided that reduce or limit such loss or its consequences.

#### A.2.1.3 *POSIX Conformance*

This really means conformance to the base standard; however, since this revision includes the core material of the Single UNIX Specification, the standard developers decided that it was appropriate to segment the conformance requirements into two, the former for the base standard, and the latter for the Single UNIX Specification.

Within POSIX.1 there are some symbolic constants that, if defined, indicate that a certain option is enabled. Other symbolic constants exist in POSIX.1 for other reasons.

As part of the revision some alignment has occurred of the options with the FIPS 151-2 profile on the POSIX.1-1990 standard. The following options from the POSIX.1-1990 standard are now mandatory:

- `_POSIX_JOB_CONTROL`
- `_POSIX_SAVED_IDS`
- `_POSIX_VDISABLE`

A POSIX-conformant system may support the XSI extensions of the Single UNIX Specification. This was intentional since the standard developers intend them to be upwards-compatible, so that a system conforming to the Single UNIX Specification can also conform to the base standard at the same time.

#### A.2.1.4 *XSI Conformance*

This section is added since the revision merges in the base volumes of the Single UNIX Specification.

XSI conformance can be thought of as a profile, selecting certain options from IEEE Std 1003.1-2001.

### A.2.1.5 Option Groups

The concept of “Option Groups” is introduced to IEEE Std 1003.1-2001 to allow collections of related functions or options to be grouped together. This has been used as follows: the “XSI Option Groups” have been created to allow super-options, collections of underlying options and related functions, to be collectively supported by XSI-conforming systems. These reflect the “Feature Groups” from the System Interfaces and Headers, Issue 5 specification.

The standard developers considered the matter of subprofiling and decided it was better to include an enabling mechanism rather than detailed normative requirements. A set of subprofiling options was developed and included later in this volume of IEEE Std 1003.1-2001 as an informative illustration.

### Subprofiling Considerations

The goal of not simultaneously fixing maximums and minimums was to allow implementations of the base standard or standards to support multiple profiles without conflict.

The following summarizes the rules for the limit types:

| Limit Type       | Fixed Value                          | Minimum Acceptable Value                            | Maximum Acceptable Value                            |
|------------------|--------------------------------------|-----------------------------------------------------|-----------------------------------------------------|
| Standard Profile | $X_s$<br>$X_p == X_s$<br>(No change) | $Y_s$<br>$Y_p \geq Y_s$<br>(May increase the limit) | $Z_s$<br>$Z_p \leq Z_s$<br>(May decrease the limit) |

The intent is that ranges specified by limits in profiles be entirely contained within the corresponding ranges of the base standard or standards being profiled, and that the unlimited end of a range in a base standard must remain unlimited in any profile of that standard.

Thus, the fixed `_POSIX_*` limits are constants and must not be changed by a profile. The variable counterparts (typically without the leading `_POSIX_`) can be changed but still remain semantically the same; that is, they still allow implementation values to vary as long as they meet the requirements for that value (be it a minimum or maximum).

Where a profile does not provide a feature upon which a limit is based, the limit is not relevant. Applications written to that profile should be written to operate independently of the value of the limit.

An example which has previously allowed implementations to support both the base standard and two other profiles in a compatible manner follows:

```
Base standard (POSIX.1-1996): _POSIX_CHILD_MAX 6
Base standard: CHILD_MAX minimum maximum _POSIX_CHILD_MAX
FIPS profile/SUSv2 CHILD_MAX 25 (minimum maximum)
```

Another example:

```
Base standard (POSIX.1-1996): _POSIX_NGROUPS_MAX 0
Base standard: NGROUPS_MAX minimum maximum _POSIX_NGROUP_MAX
FIPS profile/SUSv2 NGROUPS_MAX 8
```

A profile may lower a minimum maximum below the equivalent `_POSIX` value:

```
Base standard: _POSIX_foo_MAX Z
Base standard: foo_MAX _POSIX_foo_MAX
profile standard : foo_MAX X (X can be less than, equal to,
or greater than _POSIX_foo_MAX)
```

381 In this case an implementation conforming to the profile may not conform to the base standard,  
382 but an implementation to the base standard will conform to the profile.

#### 383 A.2.1.6 Options

384 The final subsections within *Implementation Conformance* list the core options within  
385 IEEE Std 1003.1-2001. This includes both options for the System Interfaces volume of  
386 IEEE Std 1003.1-2001 and the Shell and Utilities volume of IEEE Std 1003.1-2001.

### 387 A.2.2 Application Conformance

388 These definitions guide users or adaptors of applications in determining on which  
389 implementations an application will run and how much adaptation would be required to make  
390 it run on others. These definitions are modeled after related ones in the ISO C standard.

391 POSIX.1 occasionally uses the expressions “portable application” or “conforming application”.  
392 As they are used, these are synonyms for any of these terms. The differences between the classes  
393 of application conformance relate to the requirements for other standards, the options supported  
394 (such as the XSI extension) or, in the case of the Conforming POSIX.1 Application Using  
395 Extensions, to implementation extensions. When one of the less explicit expressions is used, it  
396 should be apparent from the context of the discussion which of the more explicit names is  
397 appropriate

#### 398 A.2.2.1 Strictly Conforming POSIX Application

399 This definition is analogous to that of an ISO C standard “conforming program”.

400 The major difference between a Strictly Conforming POSIX Application and an ISO C standard  
401 strictly conforming program is that the latter is not allowed to use features of POSIX that are not  
402 in the ISO C standard.

#### 403 A.2.2.2 Conforming POSIX Application

404 Examples of <National Bodies> include ANSI, BSI, and AFNOR.

#### 405 A.2.2.3 Conforming POSIX Application Using Extensions

406 Due to possible requirements for configuration or implementation characteristics in excess of the  
407 specifications in <limits.h> or related to the hardware (such as array size or file space), not every  
408 Conforming POSIX Application Using Extensions will run on every conforming  
409 implementation.

#### 410 A.2.2.4 Strictly Conforming XSI Application

411 This is intended to be upwards-compatible with the definition of a Strictly Conforming POSIX  
412 Application, with the addition of the facilities and functionality included in the XSI extension.

#### 413 A.2.2.5 Conforming XSI Application Using Extensions

414 Such applications may use extensions beyond the facilities defined by IEEE Std 1003.1-2001  
415 including the XSI extension, but need to document the additional requirements.

### A.2.3 Language-Dependent Services for the C Programming Language

POSIX.1 is, for historical reasons, both a specification of an operating system interface, shell and utilities, and a C binding for that specification. Efforts had been previously undertaken to generate a language-independent specification; however, that had failed, and the fact that the ISO C standard is the *de facto* primary language on POSIX and the UNIX system makes this a necessary and workable situation.

### A.2.4 Other Language-Related Specifications

There is no additional rationale provided for this section.

## A.3 Definitions

The definitions in this section are stated so that they can be used as exact substitutes for the terms in text. They should not contain requirements or cross-references to sections within IEEE Std 1003.1-2001; that is accomplished by using an informative note. In addition, the term should not be included in its own definition. Where requirements or descriptions need to be addressed but cannot be included in the definitions, due to not meeting the above criteria, these occur in the General Concepts chapter.

In this revision, the definitions have been reworked extensively to meet style requirements and to include terms from the base documents (see the Scope).

Many of these definitions are necessarily circular, and some of the terms (such as “process”) are variants of basic computing science terms that are inherently hard to define. Where some definitions are more conceptual and contain requirements, these appear in the General Concepts chapter. Those listed in this section appear in an alphabetical glossary format of terms.

Some definitions must allow extension to cover terms or facilities that are not explicitly mentioned in IEEE Std 1003.1-2001. For example, the definition of “Extended Security Controls” permits implementations beyond those defined in IEEE Std 1003.1-2001.

Some terms in the following list of notes do not appear in IEEE Std 1003.1-2001; these are marked suffixed with an asterisk (\*). Many of them have been specifically excluded from IEEE Std 1003.1-2001 because they concern system administration, implementation, or other issues that are not specific to the programming interface. Those are marked with a reason, such as “implementation-defined”.

### Appropriate Privileges

One of the fundamental security problems with many historical UNIX systems has been that the privilege mechanism is monolithic—a user has either no privileges or *all* privileges. Thus, a successful “trojan horse” attack on a privileged process defeats all security provisions. Therefore, POSIX.1 allows more granular privilege mechanisms to be defined. For many historical implementations of the UNIX system, the presence of the term “appropriate privileges” in POSIX.1 may be understood as a synonym for “superuser” (UID 0). However, other systems have emerged where this is not the case and each discrete controllable action has *appropriate privileges* associated with it. Because this mechanism is implementation-defined, it must be described in the conformance document. Although that description affects several parts of POSIX.1 where the term “appropriate privilege” is used, because the term “implementation-defined” only appears here, the description of the entire mechanism and its effects on these other sections belongs in this equivalent section of the conformance document. This is especially convenient for implementations with a single mechanism that applies in all areas, since it only needs to be described once.

**Byte**

The restriction that a byte is now exactly eight bits was a conscious decision by the standard developers. It came about due to a combination of factors, primarily the use of the type `int8_t` within the networking functions and the alignment with the ISO / IEC 9899: 1999 standard, where the `intN_t` types are now defined.

According to the ISO / IEC 9899: 1999 standard:

- The `[u]intN_t` types must be two's complement with no padding bits and no illegal values.
- All types (apart from bit fields, which are not relevant here) must occupy an integral number of bytes.
- If a type with width  $W$  occupies  $B$  bytes with  $C$  bits per byte ( $C$  is the value of `{CHAR_BIT}`), then it has  $P$  padding bits where  $P+W=B*C$ .
- Therefore, for `int8_t`  $P=0$ ,  $W=8$ . Since  $B \geq 1$ ,  $C \geq 8$ , the only solution is  $B=1$ ,  $C=8$ .

The standard developers also felt that this was not an undue restriction for the current state-of-the-art for this version of IEEE Std 1003.1, but recognize that if industry trends continue, a wider character type may be required in the future.

**Character**

The term “character” is used to mean a sequence of one or more bytes representing a single graphic symbol. The deviation in the exact text of the ISO C standard definition for “byte” meets the intent of the rationale of the ISO C standard also clears up the ambiguity raised by the term “basic execution character set”. The octet-minimum requirement is a reflection of the `{CHAR_BIT}` value.

**Clock Tick**

The ISO C standard defines a similar interval for use by the `clock()` function. There is no requirement that these intervals be the same. In historical implementations these intervals are different.

**Command**

The terms “command” and “utility” are related but have distinct meanings. Command is defined as “a directive to a shell to perform a specific task”. The directive can be in the form of a single utility name (for example, `ls`), or the directive can take the form of a compound command (for example, `"ls | grep name | pr"`). A utility is a program that can be called by name from a shell. Issuing only the name of the utility to a shell is the equivalent of a one-word command. A utility may be invoked as a separate program that executes in a different process than the command language interpreter, or it may be implemented as a part of the command language interpreter. For example, the `echo` command (the directive to perform a specific task) may be implemented such that the `echo` utility (the logic that performs the task of echoing) is in a separate program; therefore, it is executed in a process that is different from the command language interpreter. Conversely, the logic that performs the `echo` utility could be built into the command language interpreter; therefore, it could execute in the same process as the command language interpreter.

The terms “tool” and “application” can be thought of as being synonymous with “utility” from the perspective of the operating system kernel. Tools, applications, and utilities historically have run, typically, in processes above the kernel level. Tools and utilities historically have been a part of the operating system non-kernel code and have performed system-related functions, such as listing directory contents, checking file systems, repairing file systems, or extracting system

status information. Applications have not generally been a part of the operating system, and they perform non-system-related functions, such as word processing, architectural design, mechanical design, workstation publishing, or financial analysis. Utilities have most frequently been provided by the operating system distributor, applications by third-party software distributors, or by the users themselves. Nevertheless, IEEE Std 1003.1-2001 does not differentiate between tools, utilities, and applications when it comes to receiving services from the system, a shell, or the standard utilities. (For example, the *xargs* utility invokes another utility; it would be of fairly limited usefulness if the users could not run their own applications in place of the standard utilities.) Utilities are not applications in the sense that they are not themselves subject to the restrictions of IEEE Std 1003.1-2001 or any other standard—there is no requirement for *grep*, *stty*, or any of the utilities defined here to be any of the classes of conforming applications.

### Column Positions

In most 1-byte character sets, such as ASCII, the concept of column positions is identical to character positions and to bytes. Therefore, it has been historically acceptable for some implementations to describe line folding or tab stops or table column alignment in terms of bytes or character positions. Other character sets pose complications, as they can have internal representations longer than one octet and they can have display characters that have different widths on the terminal screen or printer.

In IEEE Std 1003.1-2001 the term “column positions” has been defined to mean character—not byte—positions in input files (such as “column position 7 of the FORTRAN input”). Output files describe the column position in terms of the display width of the narrowest printable character in the character set, adjusted to fit the characteristics of the output device. It is very possible that *n* column positions will not be able to hold *n* characters in some character sets, unless all of those characters are of the narrowest width. It is assumed that the implementation is aware of the width of the various characters, deriving this information from the value of *LC\_CTYPE*, and thus can determine how many column positions to allot for each character in those utilities where it is important.

The term “column position” was used instead of the more natural “column” because the latter is frequently used in the different contexts of columns of figures, columns of table values, and so on. Wherever confusion might result, these latter types of columns are referred to as “text columns”.

### Controlling Terminal

The question of which of possibly several special files referring to the terminal is meant is not addressed in POSIX.1. The filename */dev/tty* is a synonym for the controlling terminal associated with a process.

### Device Number\*

The concept is handled in *stat()* as *ID of device*.

**Direct I/O**

Historically, direct I/O refers to the system bypassing intermediate buffering, but may be extended to cover implementation-defined optimizations.

**Directory**

The format of the directory file is implementation-defined and differs radically between System V and 4.3 BSD. However, routines (derived from 4.3 BSD) for accessing directories and certain constraints on the format of the information returned by those routines are described in the `<dirent.h>` header.

**Directory Entry**

Throughout IEEE Std 1003.1-2001, the term “link” is used (about the `link()` function, for example) in describing the objects that point to files from directories.

**Display**

The Shell and Utilities volume of IEEE Std 1003.1-2001 assigns precise requirements for the terms “display” and “write”. Some historical systems have chosen to implement certain utilities without using the traditional file descriptor model. For example, the *vi* editor might employ direct screen memory updates on a personal computer, rather than a `write()` system call. An instance of user prompting might appear in a dialog box, rather than with standard error. When the Shell and Utilities volume of IEEE Std 1003.1-2001 uses the term “display”, the method of outputting to the terminal is unspecified; many historical implementations use *termcap* or *terminfo*, but this is not a requirement. The term “write” is used when the Shell and Utilities volume of IEEE Std 1003.1-2001 mandates that a file descriptor be used and that the output can be redirected. However, it is assumed that when the writing is directly to the terminal (it has not been redirected elsewhere), there is no practical way for a user or test suite to determine whether a file descriptor is being used. Therefore, the use of a file descriptor is mandated only for the redirection case and the implementation is free to use any method when the output is not redirected. The verb *write* is used almost exclusively, with the very few exceptions of those utilities where output redirection need not be supported: *tabs*, *talk*, *tput*, and *vi*.

**Dot**

The symbolic name *dot* is carefully used in POSIX.1 to distinguish the working directory filename from a period or a decimal point.

**Dot-Dot**

Historical implementations permit the use of these filenames without their special meanings. Such use precludes any meaningful use of these filenames by a Conforming POSIX.1 Application. Therefore, such use is considered an extension, the use of which makes an implementation non-conforming; see also Section A.4.11 (on page 37).



**Epoch**

Historically, the origin of UNIX system time was referred to as “00:00:00 GMT, January 1, 1970”. Greenwich Mean Time is actually not a term acknowledged by the international standards community; therefore, this term, “Epoch”, is used to abbreviate the reference to the actual standard, Coordinated Universal Time.

**FIFO Special File**

See **Pipe** (on page 24).

**File**

It is permissible for an implementation-defined file type to be non-readable or non-writable.

**File Classes**

These classes correspond to the historical sets of permission bits. The classes are general to allow implementations flexibility in expanding the access mechanism for more stringent security environments. Note that a process is in one and only one class, so there is no ambiguity.

**Filename**

At the present time, the primary responsibility for truncating filenames containing multi-byte characters must reside with the application. Some industry groups involved in internationalization believe that in the future the responsibility must reside with the kernel. For the moment, a clearer understanding of the implications of making the kernel responsible for truncation of multi-byte filenames is needed.

Character-level truncation was not adopted because there is no support in POSIX.1 that advises how the kernel distinguishes between single and multi-byte characters. Until that time, it must be incumbent upon application writers to determine where multi-byte characters must be truncated.

**File System**

Historically, the meaning of this term has been overloaded with two meanings: that of the complete file hierarchy, and that of a mountable subset of that hierarchy; that is, a mounted file system. POSIX.1 uses the term “file system” in the second sense, except that it is limited to the scope of a process (and a process’ root directory). This usage also clarifies the domain in which a file serial number is unique.

**Graphic Character**

This definition is made available for those definitions (in particular, *TZ*) which must exclude control characters.

**Group Database**

See **User Database** (on page 32).

**Group File\***

Implementation-defined; see **User Database** (on page 32).

**Historical Implementations\***

This refers to previously existing implementations of programming interfaces and operating systems that are related to the interface specified by POSIX.1.

**Hosted Implementation\***

This refers to a POSIX.1 implementation that is accomplished through interfaces from the POSIX.1 services to some alternate form of operating system kernel services. Note that the line between a hosted implementation and a native implementation is blurred, since most implementations will provide some services directly from the kernel and others through some indirect path. (For example, *fopen()* might use *open()*; or *mkfifo()* might use *mknod()*.) There is no necessary relationship between the type of implementation and its correctness, performance, and/or reliability.

**Implementation\***

This term is generally used instead of its synonym, “system”, to emphasize the consequences of decisions to be made by system implementors. Perhaps if no options or extensions to POSIX.1 were allowed, this usage would not have occurred.

The term “specific implementation” is sometimes used as a synonym for “implementation”. This should not be interpreted too narrowly; both terms can represent a relatively broad group of systems. For example, a hardware vendor could market a very wide selection of systems that all used the same instruction set, with some systems desktop models and others large multi-user minicomputers. This wide range would probably share a common POSIX.1 operating system, allowing an application compiled for one to be used on any of the others; this is a [*specific*] *implementation*. However, such a wide range of machines probably has some differences between the models. Some may have different clock rates, different file systems, different resource limits, different network connections, and so on, depending on their sizes or intended usages. Even on two identical machines, the system administrators may configure them differently. Each of these different systems is known by the term “a specific instance of a specific implementation”. This term is only used in the portions of POSIX.1 dealing with runtime queries: *sysconf()* and *pathconf()*.

**Incomplete Pathname\***

Absolute pathname has been adequately defined.

**Job Control**

In order to understand the job control facilities in POSIX.1 it is useful to understand how they are used by a job control-cognizant shell to create the user interface effect of job control.

While the job control facilities supplied by POSIX.1 can, in theory, support different types of interactive job control interfaces supplied by different types of shells, there was historically one particular interface that was most common when the standard was originally developed (provided by BSD C Shell).

This discussion describes that interface as a means of illustrating how the POSIX.1 job control facilities can be used.

Job control allows users to selectively stop (suspend) the execution of processes and continue (resume) their execution at a later point. The user typically employs this facility via the interactive interface jointly supplied by the terminal I/O driver and a command interpreter (shell).

The user can launch jobs (command pipelines) in either the foreground or background. When launched in the foreground, the shell waits for the job to complete before prompting for additional commands. When launched in the background, the shell does not wait, but immediately prompts for new commands.

If the user launches a job in the foreground and subsequently regrets this, the user can type the suspend character (typically set to <control>-Z), which causes the foreground job to stop and the shell to begin prompting for new commands. The stopped job can be continued by the user (via special shell commands) either as a foreground job or as a background job. Background jobs can also be moved into the foreground via shell commands.

If a background job attempts to access the login terminal (controlling terminal), it is stopped by the terminal driver and the shell is notified, which, in turn, notifies the user. (Terminal access includes *read()* and certain terminal control functions, and conditionally includes *write()*.) The user can continue the stopped job in the foreground, thus allowing the terminal access to succeed in an orderly fashion. After the terminal access succeeds, the user can optionally move the job into the background via the suspend character and shell commands.

#### *Implementing Job Control Shells*

The interactive interface described previously can be accomplished using the POSIX.1 job control facilities in the following way.

The key feature necessary to provide job control is a way to group processes into jobs. This grouping is necessary in order to direct signals to a single job and also to identify which job is in the foreground. (There is at most one job that is in the foreground on any controlling terminal at a time.)

The concept of process groups is used to provide this grouping. The shell places each job in a separate process group via the *setpgid()* function. To do this, the *setpgid()* function is invoked by the shell for each process in the job. It is actually useful to invoke *setpgid()* twice for each process: once in the child process, after calling *fork()* to create the process, but before calling one of the *exec* family of functions to begin execution of the program, and once in the parent shell process, after calling *fork()* to create the child. The redundant invocation avoids a race condition by ensuring that the child process is placed into the new process group before either the parent or the child relies on this being the case. The process group ID for the job is selected by the shell to be equal to the process ID of one of the processes in the job. Some shells choose to make one process in the job be the parent of the other processes in the job (if any). Other shells (for example, the C Shell) choose to make themselves the parent of all processes in the pipeline (job). In order to support this latter case, the *setpgid()* function accepts a process group ID parameter since the correct process group ID cannot be inherited from the shell. The shell itself is considered to be a job and is the sole process in its own process group.

The shell also controls which job is currently in the foreground. A foreground and background job differ in two ways: the shell waits for a foreground command to complete (or stop) before continuing to read new commands, and the terminal I/O driver inhibits terminal access by background jobs (causing the processes to stop). Thus, the shell must work cooperatively with the terminal I/O driver and have a common understanding of which job is currently in the foreground. It is the user who decides which command should be currently in the foreground, and the user informs the shell via shell commands. The shell, in turn, informs the terminal I/O driver via the *tcsetpgrp()* function. This indicates to the terminal I/O driver the process group ID

of the foreground process group (job). When the current foreground job either stops or terminates, the shell places itself in the foreground via *tcsetpgrp()* before prompting for additional commands. Note that when a job is created the new process group begins as a background process group. It requires an explicit act of the shell via *tcsetpgrp()* to move a process group (job) into the foreground.

When a process in a job stops or terminates, its parent (for example, the shell) receives synchronous notification by calling the *waitpid()* function with the WUNTRACED flag set. Asynchronous notification is also provided when the parent establishes a signal handler for SIGCHLD and does not specify the SA\_NOCLDSTOP flag. Usually all processes in a job stop as a unit since the terminal I/O driver always sends job control stop signals to all processes in the process group.

To continue a stopped job, the shell sends the SIGCONT signal to the process group of the job. In addition, if the job is being continued in the foreground, the shell invokes *tcsetpgrp()* to place the job in the foreground before sending SIGCONT. Otherwise, the shell leaves itself in the foreground and reads additional commands.

There is additional flexibility in the POSIX.1 job control facilities that allows deviations from the typical interface. Clearing the TOSTOP terminal flag allows background jobs to perform *write()* functions without stopping. The same effect can be achieved on a per-process basis by having a process set the signal action for SIGTTOU to SIG\_IGN.

Note that the terms “job” and “process group” can be used interchangeably. A login session that is not using the job control facilities can be thought of as a large collection of processes that are all in the same job (process group). Such a login session may have a partial distinction between foreground and background processes; that is, the shell may choose to wait for some processes before continuing to read new commands and may not wait for other processes. However, the terminal I/O driver will consider all these processes to be in the foreground since they are all members of the same process group.

In addition to the basic job control operations already mentioned, a job control-cognizant shell needs to perform the following actions.

When a foreground (not background) job stops, the shell must sample and remember the current terminal settings so that it can restore them later when it continues the stopped job in the foreground (via the *tcgetattr()* and *tcsetattr()* functions).

Because a shell itself can be spawned from a shell, it must take special action to ensure that subshells interact well with their parent shells.

A subshell can be spawned to perform an interactive function (prompting the terminal for commands) or a non-interactive function (reading commands from a file). When operating non-interactively, the job control shell will refrain from performing the job control-specific actions described above. It will behave as a shell that does not support job control. For example, all jobs will be left in the same process group as the shell, which itself remains in the process group established for it by its parent. This allows the shell and its children to be treated as a single job by a parent shell, and they can be affected as a unit by terminal keyboard signals.

An interactive subshell can be spawned from another job control-cognizant shell in either the foreground or background. (For example, from the C Shell, the user can execute the command, *csh &.*) Before the subshell activates job control by calling *setpgid()* to place itself in its own process group and *tcsetpgrp()* to place its new process group in the foreground, it needs to ensure that it has already been placed in the foreground by its parent. (Otherwise, there could be multiple job control shells that simultaneously attempt to control mediation of the terminal.) To determine this, the shell retrieves its own process group via *getpgrp()* and the process group of the current foreground job via *tcgetpgrp()*. If these are not equal, the shell sends SIGTTIN to

its own process group, causing itself to stop. When continued later by its parent, the shell repeats the process group check. When the process groups finally match, the shell is in the foreground and it can proceed to take control. After this point, the shell ignores all the job control stop signals so that it does not inadvertently stop itself.

#### *Implementing Job Control Applications*

Most applications do not need to be aware of job control signals and operations; the intuitively correct behavior happens by default. However, sometimes an application can inadvertently interfere with normal job control processing, or an application may choose to overtly effect job control in cooperation with normal shell procedures.

An application can inadvertently subvert job control processing by “blindly” altering the handling of signals. A common application error is to learn how many signals the system supports and to ignore or catch them all. Such an application makes the assumption that it does not know what this signal is, but knows the right handling action for it. The system may initialize the handling of job control stop signals so that they are being ignored. This allows shells that do not support job control to inherit and propagate these settings and hence to be immune to stop signals. A job control shell will set the handling to the default action and propagate this, allowing processes to stop. In doing so, the job control shell is taking responsibility for restarting the stopped applications. If an application wishes to catch the stop signals itself, it should first determine their inherited handling states. If a stop signal is being ignored, the application should continue to ignore it. This is directly analogous to the recommended handling of SIGINT described in the referenced UNIX Programmer’s Manual.

If an application is reading the terminal and has disabled the interpretation of special characters (by clearing the ISIG flag), the terminal I/O driver will not send SIGTSTP when the suspend character is typed. Such an application can simulate the effect of the suspend character by recognizing it and sending SIGTSTP to its process group as the terminal driver would have done. Note that the signal is sent to the process group, not just to the application itself; this ensures that other processes in the job also stop. (Note also that other processes in the job could be children, siblings, or even ancestors.) Applications should not assume that the suspend character is <control>-Z (or any particular value); they should retrieve the current setting at startup.

#### *Implementing Job Control Systems*

The intent in adding 4.2 BSD-style job control functionality was to adopt the necessary 4.2 BSD programmatic interface with only minimal changes to resolve syntactic or semantic conflicts with System V or to close recognized security holes. The goal was to maximize the ease of providing both conforming implementations and Conforming POSIX.1 Applications.

It is only useful for a process to be affected by job control signals if it is the descendant of a job control shell. Otherwise, there will be nothing that continues the stopped process.

POSIX.1 does not specify how controlling terminal access is affected by a user logging out (that is, by a controlling process terminating). 4.2 BSD uses the *vhangup()* function to prevent any access to the controlling terminal through file descriptors opened prior to logout. System V does not prevent controlling terminal access through file descriptors opened prior to logout (except for the case of the special file, */dev/tty*). Some implementations choose to make processes immune from job control after logout (that is, such processes are always treated as if in the foreground); other implementations continue to enforce foreground/background checks after logout. Therefore, a Conforming POSIX.1 Application should not attempt to access the controlling terminal after logout since such access is unreliable. If an implementation chooses to deny access to a controlling terminal after its controlling process exits, POSIX.1 requires a certain type of behavior (see **Controlling Terminal** (on page 15)).

**796 Kernel\***

797 See **System Call\*** (on page 30).

**798 Library Routine\***

799 See **System Call\*** (on page 30).

**800 Logical Device\***

801 Implementation-defined.

**802 Map**

803 The definition of map is included to clarify the usage of mapped pages in the description of the  
804 behavior of process memory locking.

**805 Memory-Resident**

806 The term “memory-resident” is historically understood to mean that the so-called resident  
807 pages are actually present in the physical memory of the computer system and are immune from  
808 swapping, paging, copy-on-write faults, and so on. This is the actual intent of  
809 IEEE Std 1003.1-2001 in the process memory locking section for implementations where this is  
810 logical. But for some implementations—primarily mainframes—actually locking pages into  
811 primary storage is not advantageous to other system objectives, such as maximizing throughput.  
812 For such implementations, memory locking is a “hint” to the implementation that the  
813 application wishes to avoid situations that would cause long latencies in accessing memory.  
814 Furthermore, there are other implementation-defined issues with minimizing memory access  
815 latencies that “memory residency” does not address—such as MMU reload faults. The definition  
816 attempts to accommodate various implementations while allowing conforming applications to  
817 specify to the implementation that they want or need the best memory access times that the  
818 implementation can provide.

**819 Memory Object\***

820 The term “memory object” usually implies shared memory. If the object is the same as a  
821 filename in the file system name space of the implementation, it is expected that the data written  
822 into the memory object be preserved on disk. A memory object may also apply to a physical  
823 device on an implementation. In this case, writes to the memory object are sent to the controller  
824 for the device and reads result in control registers being returned.

**825 Mount Point\***

826 The directory on which a “mounted file system” is mounted. This term, like *mount()* and  
827 *umount()*, was not included because it was implementation-defined.

**828 Mounted File System\***

829 See **File System** (on page 17).

**Name**

There are no explicit limits in IEEE Std 1003.1-2001 on the sizes of names, words (see the definition of word in the Base Definitions volume of IEEE Std 1003.1-2001), lines, or other objects. However, other implicit limits do apply: shell script lines produced by many of the standard utilities cannot exceed {LINE\_MAX} and the sum of exported variables comes under the {ARG\_MAX} limit. Historical shells dynamically allocate memory for names and words and parse incoming lines a character at a time. Lines cannot have an arbitrary {LINE\_MAX} limit because of historical practice, such as makefiles, where *make* removes the <newline>s associated with the commands for a target and presents the shell with one very long line. The text on INPUT FILES in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 1.11, Utility Description Defaults does allow a shell to run out of memory, but it cannot have arbitrary programming limits.

**Native Implementation\***

This refers to an implementation of POSIX.1 that interfaces directly to an operating system kernel; see also *hosted implementation*. A similar concept is a native UNIX system, which would be a kernel derived from one of the original UNIX system products.

**Nice Value**

This definition is not intended to suggest that all processes in a system have priorities that are comparable. Scheduling policy extensions, such as adding realtime priorities, make the notion of a single underlying priority for all scheduling policies problematic. Some implementations may implement the features related to *nice* to affect all processes on the system, others to affect just the general time-sharing activities implied by IEEE Std 1003.1-2001, and others may have no effect at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies is possible.

**Open File Description**

An “open file description”, as it is currently named, describes how a file is being accessed. What is currently called a “file descriptor” is actually just an identifier or “handle”; it does not actually describe anything.

The following alternate names were discussed:

- For “open file description”:  
“open instance”, “file access description”, “open file information”, and “file access information”.
- For “file descriptor”:  
“file handle”, “file number” (cf., *fileno()*). Some historical implementations use the term “file table entry”.

**Orphaned Process Group**

Historical implementations have a concept of an orphaned process, which is a process whose parent process has exited. When job control is in use, it is necessary to prevent processes from being stopped in response to interactions with the terminal after they no longer are controlled by a job control-cognizant program. Because signals generated by the terminal are sent to a process group and not to individual processes, and because a signal may be provoked by a process that is not orphaned, but sent to another process that is orphaned, it is necessary to define an orphaned process group. The definition assumes that a process group will be manipulated as a group and that the job control-cognizant process controlling the group is outside of the group

and is the parent of at least one process in the group (so that state changes may be reported via `waitpid()`). Therefore, a group is considered to be controlled as long as at least one process in the group has a parent that is outside of the process group, but within the session.

This definition of orphaned process groups ensures that a session leader's process group is always considered to be orphaned, and thus it is prevented from stopping in response to terminal signals.

## **Page**

The term “page” is defined to support the description of the behavior of memory mapping for shared memory and memory mapped files, and the description of the behavior of process memory locking. It is not intended to imply that shared memory/file mapping and memory locking are applicable only to “paged” architectures. For the purposes of IEEE Std 1003.1-2001, whatever the granularity on which an architecture supports mapping or locking, this is considered to be a “page”. If an architecture cannot support the memory mapping or locking functions specified by IEEE Std 1003.1-2001 on any granularity, then these options will not be implemented on the architecture.

## **Passwd File\***

Implementation-defined; see **User Database** (on page 32).

## **Parent Directory**

There may be more than one directory entry pointing to a given directory in some implementations. The wording here identifies that exactly one of those is the parent directory. In pathname resolution, dot-dot is identified as the way that the unique directory is identified. (That is, the parent directory is the one to which dot-dot points.) In the case of a remote file system, if the same file system is mounted several times, it would appear as if they were distinct file systems (with interesting synchronization properties).

## **Pipe**

It proved convenient to define a pipe as a special case of a FIFO, even though historically the latter was not introduced until System III and does not exist at all in 4.3 BSD.

## **Portable Filename Character Set**

The encoding of this character set is not specified—specifically, ASCII is not required. But the implementation must provide a unique character code for each of the printable graphics specified by POSIX.1; see also Section A.4.6 (on page 34).

Situations where characters beyond the portable filename character set (or historically ASCII or the ISO/IEC 646:1991 standard) would be used (in a context where the portable filename character set or the ISO/IEC 646:1991 standard is required by POSIX.1) are expected to be common. Although such a situation renders the use technically non-compliant, mutual agreement among the users of an extended character set will make such use portable between those users. Such a mutual agreement could be formalized as an optional extension to POSIX.1. (Making it required would eliminate too many possible systems, as even those systems using the ISO/IEC 646:1991 standard as a base character set extend their character sets for Western Europe and the rest of the world in different ways.)

Nothing in POSIX.1 is intended to preclude the use of extended characters where interchange is not required or where mutual agreement is obtained. It has been suggested that in several places “should” be used instead of “shall”. Because (in the worst case) use of any character beyond the



portable filename character set would render the program or data not portable to all possible systems, no extensions are permitted in this context.

### Regular File

POSIX.1 does not intend to preclude the addition of structuring data (for example, record lengths) in the file, as long as such data is not visible to an application that uses the features described in POSIX.1.

### Root Directory

This definition permits the operation of *chroot()*, even though that function is not in POSIX.1; see also Section A.4.5 (on page 33).

### Root File System\*

Implementation-defined.

### Root of a File System\*

Implementation-defined; see **Mount Point\*** (on page 22).

### Signal

The definition implies a double meaning for the term. Although a signal is an event, common usage implies that a signal is an identifier of the class of event.

### Superuser\*

This concept, with great historical significance to UNIX system users, has been replaced with the notion of appropriate privileges.

### Supplementary Group ID

The POSIX.1-1990 standard is inconsistent in its treatment of supplementary groups. The definition of supplementary group ID explicitly permits the effective group ID to be included in the set, but wording in the description of the *setuid()* and *setgid()* functions states: “Any supplementary group IDs of the calling process remain unchanged by these function calls”. In the case of *setgid()* this contradicts that definition. In addition, some felt that the unspecified behavior in the definition of supplementary group IDs adds unnecessary portability problems. The standard developers considered several solutions to this problem:

1. Reword the description of *setgid()* to permit it to change the supplementary group IDs to reflect the new effective group ID. A problem with this is that it adds more “may”s to the wording and does not address the portability problems of this optional behavior.
2. Mandate the inclusion of the effective group ID in the supplementary set (giving {NGROUPS\_MAX} a minimum value of 1). This is the behavior of 4.4 BSD. In that system, the effective group ID is the first element of the array of supplementary group IDs (there is no separate copy stored, and changes to the effective group ID are made only in the supplementary group set). By convention, the initial value of the effective group ID is duplicated elsewhere in the array so that the initial value is not lost when executing a set-group-ID program.
3. Change the definition of supplementary group ID to exclude the effective group ID and specify that the effective group ID does not change the set of supplementary group IDs. This is the behavior of 4.2 BSD, 4.3 BSD, and System V Release 4.

4. Change the definition of supplementary group ID to exclude the effective group ID, and require that *getgroups()* return the union of the effective group ID and the supplementary group IDs.
5. Change the definition of {NGROUPS\_MAX} to be one more than the number of supplementary group IDs, so it continues to be the number of values returned by *getgroups()* and existing applications continue to work. This alternative is effectively the same as the second (and might actually have the same implementation).

The standard developers decided to permit either 2 or 3. The effective group ID is orthogonal to the set of supplementary group IDs, and it is implementation-defined whether *getgroups()* returns this. If the effective group ID is returned with the set of supplementary group IDs, then all changes to the effective group ID affect the supplementary group set returned by *getgroups()*. It is permissible to eliminate duplicates from the list returned by *getgroups()*. However, if a group ID is contained in the set of supplementary group IDs, setting the group ID to that value and then to a different value should not remove that value from the supplementary group IDs.

The definition of supplementary group IDs has been changed to not include the effective group ID. This simplifies permanent rationale and makes the relevant functions easier to understand. The *getgroups()* function has been modified so that it can, on an implementation-defined basis, return the effective group ID. By making this change, functions that modify the effective group ID do not need to discuss adding to the supplementary group list; the only view into the supplementary group list that the application writer has is through the *getgroups()* function.

## Symbolic Link

Many implementations associate no attributes, including ownership with symbolic links. Security experts encouraged consideration for defining these attributes as optional. Consideration was given to changing *utime()* to allow modification of the times for a symbolic link, or as an alternative adding an *lutime()* interface. Modifications to *chown()* were also considered: allow changing symbolic link ownership or alternatively adding *lchown()*. As a result of alignment with the Single UNIX Specification, the *lchown()* function is included as part of the XSI extension and XSI-conformant systems may support an owner and a group associated with a symbolic link. There was no consensus to define further attributes for symbolic links, and for systems not supporting the XSI extension only the file type bits in the *st\_mode* member and the *st\_size* member of the **stat** structure are required to be applicable to symbolic links.

Historical implementations were followed when determining which interfaces should apply to symbolic links. Interfaces that historically followed symbolic links include *chmod()*, *link()*, and *utime()*. Interfaces that historically do not follow symbolic links include *chown()*, *lstat()*, *readlink()*, *rename()*, *remove()*, *rmdir()*, and *unlink()*. IEEE Std 1003.1-2001 deviates from historical practice only in the case of *chown()*. Because there is no requirement for systems not supporting the XSI extension that there is an association of ownership with symbolic links, there was no interface in the base standard to change ownership. In addition, other implementations of symbolic links have modified *chown()* to follow symbolic links.

In the case of symbolic links, IEEE Std 1003.1-2001 states that a trailing slash is considered to be the final component of a pathname rather than the pathname component that preceded it. This is the behavior of historical implementations. For example, for */a/b* and */a/b/*, if */a/b* is a symbolic link to a directory, then */a/b* refers to the symbolic link, and */a/b/* is the same as */a/b/.*, which is the directory to which the symbolic link points.

For multi-level security purposes, it is possible to have the link read mode govern permission for the *readlink()* function. It is also possible that the read permissions of the directory containing the link be used for this purpose. Implementations may choose to use either of these methods; however, this is not current practice and neither method is specified.

Several reasons were advanced for requiring that when a symbolic link is used as the source argument to the *link()* function, the resulting link will apply to the file named by the contents of the symbolic link rather than to the symbolic link itself. This is the case in historical implementations. This action was preferred, as it supported the traditional idea of persistence with respect to the target of a hard link. This decision is appropriate in light of a previous decision not to require association of attributes with symbolic links, thereby allowing implementations which do not use inodes. Opposition centered on the lack of symmetry on the part of the *link()* and *unlink()* function pair with respect to symbolic links.

Because a symbolic link and its referenced object coexist in the file system name space, confusion can arise in distinguishing between the link itself and the referenced object. Historically, utilities and system calls have adopted their own link following conventions in a somewhat *ad hoc* fashion. Rules for a uniform approach are outlined here, although historical practice has been adhered to as much as was possible. To promote consistent system use, user-written utilities are encouraged to follow these same rules.

Symbolic links are handled either by operating on the link itself, or by operating on the object referenced by the link. In the latter case, an application or system call is said to “follow” the link. Symbolic links may reference other symbolic links, in which case links are dereferenced until an object that is not a symbolic link is found, a symbolic link that references a file that does not exist is found, or a loop is detected. (Current implementations do not detect loops, but have a limit on the number of symbolic links that they will dereference before declaring it an error.)

There are four domains for which default symbolic link policy is established in a system. In almost all cases, there are utility options that override this default behavior. The four domains are as follows:

1. Symbolic links specified to system calls that take filename arguments
2. Symbolic links specified as command line filename arguments to utilities that are not performing a traversal of a file hierarchy
3. Symbolic links referencing files not of type directory, specified to utilities that are performing a traversal of a file hierarchy
4. Symbolic links referencing files of type directory, specified to utilities that are performing a traversal of a file hierarchy

#### *First Domain*

The first domain is considered in earlier rationale.

#### *Second Domain*

The reason this category is restricted to utilities that are not traversing the file hierarchy is that some standard utilities take an option that specifies a hierarchical traversal, but by default operate on the arguments themselves. Generally, users specifying the option for a file hierarchy traversal wish to operate on a single, physical hierarchy, and therefore symbolic links, which may reference files outside of the hierarchy, are ignored. For example, *chown owner file* is a different operation from the same command with the **-R** option specified. In this example, the behavior of the command *chown owner file* is described here, while the behavior of the command *chown -R owner file* is described in the third and fourth domains.

The general rule is that the utilities in this category follow symbolic links named as arguments.

Exceptions in the second domain are:

- The *mv* and *rm* utilities do not follow symbolic links named as arguments, but respectively attempt to rename or delete them.

- The *ls* utility is also an exception to this rule. For compatibility with historical systems, when the **-R** option is not specified, the *ls* utility follows symbolic links named as arguments if the **-L** option is specified or if the **-F**, **-d**, or **-l** options are not specified. (If the **-L** option is specified, *ls* always follows symbolic links; it is the only utility where the **-L** option affects its behavior even though a tree walk is not being performed.)

All other standard utilities, when not traversing a file hierarchy, always follow symbolic links named as arguments.

Historical practice is that the **-h** option is specified if standard utilities are to act upon symbolic links instead of upon their targets. Examples of commands that have historically had a **-h** option for this purpose are the *chgrp*, *chown*, *file*, and *test* utilities.

### Third Domain

The third domain is symbolic links, referencing files not of type directory, specified to utilities that are performing a traversal of a file hierarchy. (This includes symbolic links specified as command line filename arguments or encountered during the traversal.)

The intention of the Shell and Utilities volume of IEEE Std 1003.1-2001 is that the operation that the utility is performing is applied to the symbolic link itself, if that operation is applicable to symbolic links. The reason that the operation is not required is that symbolic links in some implementations do not have such attributes as a file owner, and therefore the *chown* operation would be meaningless. If symbolic links on the system have an owner, it is the intention that the utility *chown* cause the owner of the symbolic link to change. If symbolic links do not have an owner, the symbolic link should be ignored. Specifically, by default, no change should be made to the file referenced by the symbolic link.

### Fourth Domain

The fourth domain is symbolic links referencing files of type directory, specified to utilities that are performing a traversal of a file hierarchy. (This includes symbolic links specified as command line filename arguments or encountered during the traversal.)

Most standard utilities do not, by default, indirect into the file hierarchy referenced by the symbolic link. (The Shell and Utilities volume of IEEE Std 1003.1-2001 uses the informal term “physical walk” to describe this case. The case where the utility does indirect through the symbolic link is termed a “logical walk”.)

There are three reasons for the default to be a physical walk:

1. With very few exceptions, a physical walk has been the historical default on UNIX systems supporting symbolic links. Because some utilities (that is, *rm*) must default to a physical walk, regardless, changing historical practice in this regard would be confusing to users and needlessly incompatible.
2. For systems where symbolic links have the historical file attributes (that is, *owner*, *group*, *mode*), defaulting to a logical traversal would require the addition of a new option to the commands to modify the attributes of the link itself. This is painful and more complex than the alternatives.
3. There is a security issue with defaulting to a logical walk. Historically, the command *chown -R user file* has been safe for the superuser because *setuid* and *setgid* bits were lost when the ownership of the file was changed. If the walk were logical, changing ownership would no longer be safe because a user might have inserted a symbolic link pointing to any file in the tree. Again, this would necessitate the addition of an option to the commands doing hierarchy traversal to not indirect through the symbolic links, and historical scripts doing recursive walks would instantly become security problems. While this is mostly an

1096 issue for system administrators, it is preferable to not have different defaults for different  
1097 classes of users.

1098 However, the standard developers agreed to leave it unspecified to achieve consensus.

1099 As consistently as possible, users may cause standard utilities performing a file hierarchy  
1100 traversal to follow any symbolic links named on the command line, regardless of the type of file  
1101 they reference, by specifying the **-H** (for half logical) option. This option is intended to make the  
1102 command line name space look like the logical name space.

1103 As consistently as possible, users may cause standard utilities performing a file hierarchy  
1104 traversal to follow any symbolic links named on the command line as well as any symbolic links  
1105 encountered during the traversal, regardless of the type of file they reference, by specifying the  
1106 **-L** (for logical) option. This option is intended to make the entire name space look like the  
1107 logical name space.

1108 For consistency, implementors are encouraged to use the **-P** (for “physical”) flag to specify the  
1109 physical walk in utilities that do logical walks by default for whatever reason. The only standard  
1110 utilities that require the **-P** option are *cd* and *pwd*; see the note below.

1111 When one or more of the **-H**, **-L**, and **-P** flags can be specified, the last one specified determines  
1112 the behavior of the utility. This permits users to alias commands so that the default behavior is a  
1113 logical walk and then override that behavior on the command line.

#### 1114 *Exceptions in the Third and Fourth Domains*

1115 The *ls* and *rm* utilities are exceptions to these rules. The *rm* utility never follows symbolic links  
1116 and does not support the **-H**, **-L**, or **-P** options. Some historical versions of *ls* always followed  
1117 symbolic links given on the command line whether the **-L** option was specified or not. Historical  
1118 versions of *ls* did not support the **-H** option. In IEEE Std 1003.1-2001, unless one of the **-H** or **-L**  
1119 options is specified, the *ls* utility only follows symbolic links to directories that are given as  
1120 operands. The *ls* utility does not support the **-P** option.

1121 The Shell and Utilities volume of IEEE Std 1003.1-2001 requires that the standard utilities *ls*, *find*,  
1122 and *pax* detect infinite loops when doing logical walks; that is, a directory, or more commonly a  
1123 symbolic link, that refers to an ancestor in the current file hierarchy. If the file system itself is  
1124 corrupted, causing the infinite loop, it may be impossible to recover. Because *find* and *ls* are often  
1125 used in system administration and security applications, they should attempt to recover and  
1126 continue as best as they can. The *pax* utility should terminate because the archive it was creating  
1127 is by definition corrupted. Other, less vital, utilities should probably simply terminate as well.  
1128 Implementations are strongly encouraged to detect infinite loops in all utilities.

1129 Historical practice is shown in Table A-1 (on page 30). The heading **SVID3** stands for the Third  
1130 Edition of the System V Interface Definition.

1131 Historically, several shells have had built-in versions of the *pwd* utility. In some of these shells,  
1132 *pwd* reported the physical path, and in others, the logical path. Implementations of the shell  
1133 corresponding to IEEE Std 1003.1-2001 must report the logical path by default. Earlier versions of  
1134 IEEE Std 1003.1-2001 did not require the *pwd* utility to be a built-in utility. Now that *pwd* is  
1135 required to set an environment variable in the current shell execution environment, it must be a  
1136 built-in utility.

1137 The *cd* command is required, by default, to treat the filename dot-dot logically. Implementors are  
1138 required to support the **-P** flag in *cd* so that users can have their current environment handled  
1139 physically. In 4.3 BSD, *chgrp* during tree traversal changed the group of the symbolic link, not  
1140 the target. Symbolic links in 4.4 BSD do not have *owner*, *group*, *mode*, or other standard UNIX  
1141 system file attributes.

1142

**Table A-1** Historical Practice for Symbolic Links

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

| Utility      | SVID3          | 4.3 BSD   | 4.4 BSD        | POSIX     | Comments                                |
|--------------|----------------|-----------|----------------|-----------|-----------------------------------------|
| <i>cd</i>    |                |           |                | <b>-L</b> | Treat ". ." logically.                  |
| <i>cd</i>    |                |           |                | <b>-P</b> | Treat ". ." physically.                 |
| <i>chgrp</i> |                |           | <b>-H</b>      | <b>-H</b> | Follow command line symlinks.           |
| <i>chgrp</i> |                |           | <b>-h</b>      | <b>-L</b> | Follow symlinks.                        |
| <i>chgrp</i> | <b>-h</b>      |           |                | <b>-h</b> | Affect the symlink.                     |
| <i>chmod</i> |                |           |                |           | Affect the symlink.                     |
| <i>chmod</i> |                |           | <b>-H</b>      |           | Follow command line symlinks.           |
| <i>chmod</i> |                |           | <b>-h</b>      |           | Follow symlinks.                        |
| <i>chown</i> |                |           | <b>-H</b>      | <b>-H</b> | Follow command line symlinks.           |
| <i>chown</i> |                |           | <b>-h</b>      | <b>-L</b> | Follow symlinks.                        |
| <i>chown</i> | <b>-h</b>      |           |                | <b>-h</b> | Affect the symlink.                     |
| <i>cp</i>    |                |           | <b>-H</b>      | <b>-H</b> | Follow command line symlinks.           |
| <i>cp</i>    |                |           | <b>-h</b>      | <b>-L</b> | Follow symlinks.                        |
| <i>cpio</i>  | <b>-L</b>      |           | <b>-L</b>      |           | Follow symlinks.                        |
| <i>du</i>    |                |           | <b>-H</b>      | <b>-H</b> | Follow command line symlinks.           |
| <i>du</i>    |                |           | <b>-h</b>      | <b>-L</b> | Follow symlinks.                        |
| <i>file</i>  | <b>-h</b>      |           |                | <b>-h</b> | Affect the symlink.                     |
| <i>find</i>  |                |           | <b>-H</b>      | <b>-H</b> | Follow command line symlinks.           |
| <i>find</i>  |                |           | <b>-h</b>      | <b>-L</b> | Follow symlinks.                        |
| <i>find</i>  | <b>-follow</b> |           | <b>-follow</b> |           | Follow symlinks.                        |
| <i>ln</i>    | <b>-s</b>      | <b>-s</b> | <b>-s</b>      | <b>-s</b> | Create a symbolic link.                 |
| <i>ls</i>    | <b>-L</b>      | <b>-L</b> | <b>-L</b>      | <b>-L</b> | Follow symlinks.                        |
| <i>ls</i>    |                |           |                | <b>-H</b> | Follow command line symlinks.           |
| <i>mv</i>    |                |           |                |           | Operates on the symlink.                |
| <i>pax</i>   |                |           | <b>-H</b>      | <b>-H</b> | Follow command line symlinks.           |
| <i>pax</i>   |                |           | <b>-h</b>      | <b>-L</b> | Follow symlinks.                        |
| <i>pwd</i>   |                |           |                | <b>-L</b> | Printed path may contain symlinks.      |
| <i>pwd</i>   |                |           |                | <b>-P</b> | Printed path will not contain symlinks. |
| <i>rm</i>    |                |           |                |           | Operates on the symlink.                |
| <i>tar</i>   |                |           | <b>-H</b>      |           | Follow command line symlinks.           |
| <i>tar</i>   |                | <b>-h</b> | <b>-h</b>      |           | Follow symlinks.                        |
| <i>test</i>  | <b>-h</b>      |           | <b>-h</b>      | <b>-h</b> | Affect the symlink.                     |

1176

**Synchronously-Generated Signal**

1177

1178

Those signals that may be generated synchronously include SIGABRT, SIGBUS, SIGILL, SIGFPE, SIGPIPE, and SIGSEGV.

1179

1180

Any signal sent via the *raise()* function or a *kill()* function targeting the current process is also considered synchronous.

1181

**System Call\***

1182

1183

The distinction between a “system call” and a “library routine” is an implementation detail that may differ between implementations and has thus been excluded from POSIX.1.

1184

See “Interface, Not Implementation” in **Introduction** (on page xiii).

**System Reboot**

A “system reboot” is an event initiated by an unspecified circumstance that causes all processes (other than special system processes) to be terminated in an implementation-defined manner, after which any changes to the state and contents of files created or written to by a Conforming POSIX.1 Application prior to the event are implementation-defined.

**Synchronized I/O Data (and File) Integrity Completion**

These terms specify that for synchronized read operations, pending writes must be successfully completed before the read operation can complete. This is motivated by two circumstances. Firstly, when synchronizing processes can access the same file, but not share common buffers (such as for a remote file system), this requirement permits the reading process to guarantee that it can read data written remotely. Secondly, having data written synchronously is insufficient to guarantee the order with respect to a subsequent write by a reading process, and thus this extra read semantic is necessary.

**Text File**

The term “text file” does not prevent the inclusion of control or other non-printable characters (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either able to process the special characters or they explicitly describe their limitations within their individual descriptions. The definition of “text file” has caused controversy. The only difference between text and binary files is that text files have lines of less than {LINE\_MAX} bytes, with no NUL characters, each terminated by a <newline>. The definition allows a file with a single <newline>, but not a totally empty file, to be called a text file. If a file ends with an incomplete line it is not strictly a text file by this definition. The <newline> referred to in IEEE Std 1003.1-2001 is not some generic line separator, but a single character; files created on systems where they use multiple characters for ends of lines are not portable to all conforming systems without some translation process unspecified by IEEE Std 1003.1-2001.

**Thread**

IEEE Std 1003.1-2001 defines a thread to be a flow of control within a process. Each thread has a minimal amount of private state; most of the state associated with a process is shared among all of the threads in the process. While most multi-thread extensions to POSIX have taken this approach, others have made different decisions.

**Note:** The choice to put threads within a process does not constrain implementations to implement threads in that manner. However, all functions have to behave as though threads share the indicated state information with the process from which they were created.

Threads need to share resources in order to cooperate. Memory has to be widely shared between threads in order for the threads to cooperate at a fine level of granularity. Threads keep data structures and the locks protecting those data structures in shared memory. For a data structure to be usefully shared between threads, such structures should not refer to any data that can only be interpreted meaningfully by a single thread. Thus, any system resources that might be referred to in data structures need to be shared between all threads. File descriptors, pathnames, and pointers to stack variables are all things that programmers want to share between their threads. Thus, the file descriptor table, the root directory, the current working directory, and the address space have to be shared.

Library implementations are possible as long as the effective behavior is as if system services invoked by one thread do not suspend other threads. This may be difficult for some library implementations on systems that do not provide asynchronous facilities.

1230 See Section B.2.9 (on page 150) for additional rationale.

### 1231 **Thread ID**

1232 See Section B.2.9.2 (on page 166) for additional rationale.

### 1233 **Thread-Safe Function**

1234 All functions required by IEEE Std 1003.1-2001 need to be thread-safe; see Section A.4.16 (on  
1235 page 40) and Section B.2.9.1 (on page 163) for additional rationale.

### 1236 **User Database**

1237 There are no references in IEEE Std 1003.1-2001 to a “passwd file” or a “group file”, and there is  
1238 no requirement that the *group* or *passwd* databases be kept in files containing editable text. Many  
1239 large timesharing systems use *passwd* databases that are hashed for speed. Certain security  
1240 classifications prohibit certain information in the *passwd* database from being publicly readable.

1241 The term “encoded” is used instead of “encrypted” in order to avoid the implementation  
1242 connotations (such as reversibility or use of a particular algorithm) of the latter term.

1243 The *getgrent()*, *setgrent()*, *endgrent()*, *getpwent()*, *setpwent()*, and *endpwent()* functions are not  
1244 included as part of the base standard because they provide a linear database search capability  
1245 that is not generally useful (the *getpwuid()*, *getpwnam()*, *getgrgid()*, and *getgrnam()* functions are  
1246 provided for keyed lookup) and because in certain distributed systems, especially those with  
1247 different authentication domains, it may not be possible or desirable to provide an application  
1248 with the ability to browse the system databases indiscriminately. They are provided on XSI-  
1249 conformant systems due to their historical usage by many existing applications.

1250 A change from historical implementations is that the structures used by these functions have  
1251 fields of the types **gid\_t** and **uid\_t**, which are required to be defined in the **<sys/types.h>** header.  
1252 IEEE Std 1003.1-2001 requires implementations to ensure that these types are defined by  
1253 inclusion of **<grp.h>** and **<pwd.h>**, respectively, without imposing any name space pollution or  
1254 errors from redefinition of types.

1255 IEEE Std 1003.1-2001 is silent about the content of the strings containing user or group names.  
1256 These could be digit strings. IEEE Std 1003.1-2001 is also silent as to whether such digit strings  
1257 bear any relationship to the corresponding (numeric) user or group ID.

### 1258 **Database Access**

1259 The thread-safe versions of the user and group database access functions return values in user-  
1260 supplied buffers instead of possibly using static data areas that may be overwritten by each call.

### 1261 **Virtual Processor\***

1262 The term “virtual processor” was chosen as a neutral term describing all kernel-level  
1263 schedulable entities, such as processes, Mach tasks, or lightweight processes. Implementing  
1264 threads using multiple processes as virtual processors, or implementing multiplexed threads  
1265 above a virtual processor layer, should be possible, provided some mechanism has also been  
1266 implemented for sharing state between processes or virtual processors. Many systems may also  
1267 wish to provide implementations of threads on systems providing “shared processes” or  
1268 “variable-weight processes”. It was felt that exposing such implementation details would  
1269 severely limit the type of systems upon which the threads interface could be supported and  
1270 prevent certain types of valid implementations. It was also determined that a virtual processor  
1271 interface was out of the scope of the Rationale (Informative) volume of IEEE Std 1003.1-2001.



1272 **XSI**

1273 This is introduced to allow IEEE Std 1003.1-2001 to be adopted as an IEEE standard and an Open  
 1274 Group Technical Standard, serving both the POSIX and the Single UNIX Specification in a core  
 1275 set of volumes.

1276 The term “XSI” has been used for 10 years in connection with the XPG series and the first and  
 1277 second versions of the base volumes of the Single UNIX Specification. The XSI margin code was  
 1278 introduced to denote the extended or more restrictive semantics beyond POSIX that are  
 1279 applicable to UNIX systems.

1280 **A.4 General Concepts**1281 **A.4.1 Concurrent Execution**

1282 There is no additional rationale provided for this section.

1283 **A.4.2 Directory Protection**

1284 There is no additional rationale provided for this section.

1285 **A.4.3 Extended Security Controls**

1286 Allowing an implementation to define extended security controls enables the use of  
 1287 IEEE Std 1003.1-2001 in environments that require different or more rigorous security than that  
 1288 provided in POSIX.1. Extensions are allowed in two areas: privilege and file access permissions.  
 1289 The semantics of these areas have been defined to permit extensions with reasonable, but not  
 1290 exact, compatibility with all existing practices. For example, the elimination of the superuser  
 1291 definition precludes identifying a process as privileged or not by virtue of its effective user ID.

1292 **A.4.4 File Access Permissions**

1293 A process should not try to anticipate the result of an attempt to access data by *a priori* use of  
 1294 these rules. Rather, it should make the attempt to access data and examine the return value (and  
 1295 possibly *errno* as well), or use *access()*. An implementation may include other security  
 1296 mechanisms in addition to those specified in POSIX.1, and an access attempt may fail because of  
 1297 those additional mechanisms, even though it would succeed according to the rules given in this  
 1298 section. (For example, the user’s security level might be lower than that of the object of the access  
 1299 attempt.) The supplementary group IDs provide another reason for a process to not attempt to  
 1300 anticipate the result of an access attempt.

1301 **A.4.5 File Hierarchy**

1302 Though the file hierarchy is commonly regarded to be a tree, POSIX.1 does not define it as such  
 1303 for three reasons:

- 1304 1. Links may join branches.
- 1305 2. In some network implementations, there may be no single absolute root directory; see  
 1306 *pathname resolution*.
- 1307 3. With symbolic links, the file system need not be a tree or even a directed acyclic graph.

1308 **A.4.6 Filenames**

1309 Historically, certain filenames have been reserved. This list includes **core**, **/etc/passwd**, and so  
 1310 on. Conforming applications should avoid these.

1311 Most historical implementations prohibit case folding in filenames; that is, treating uppercase  
 1312 and lowercase alphabetic characters as identical. However, some consider case folding desirable:

- 1313 • For user convenience
- 1314 • For ease-of-implementation of the POSIX.1 interface as a hosted system on some popular  
 1315 operating systems

1316 Variants, such as maintaining case distinctions in filenames, but ignoring them in comparisons,  
 1317 have been suggested. Methods of allowing escaped characters of the case opposite the default  
 1318 have been proposed.

1319 Many reasons have been expressed for not allowing case folding, including:

- 1320 • No solid evidence has been produced as to whether case-sensitivity or case-insensitivity is  
 1321 more convenient for users.
- 1322 • Making case-insensitivity a POSIX.1 implementation option would be worse than either  
 1323 having it or not having it, because:
  - 1324 — More confusion would be caused among users.
  - 1325 — Application developers would have to account for both cases in their code.
  - 1326 — POSIX.1 implementors would still have other problems with native file systems, such as  
 1327 short or otherwise constrained filenames or pathnames, and the lack of hierarchical  
 1328 directory structure.
- 1329 • Case folding is not easily defined in many European languages, both because many of them  
 1330 use characters outside the US ASCII alphabetic set, and because:
  - 1331 — In Spanish, the digraph "ll" is considered to be a single letter, the capitalized form of  
 1332 which may be either "Ll" or "LL", depending on context.
  - 1333 — In French, the capitalized form of a letter with an accent may or may not retain the accent,  
 1334 depending on the country in which it is written.
  - 1335 — In German, the sharp ess may be represented as a single character resembling a Greek  
 1336 beta (ß) in lowercase, but as the digraph "SS" in uppercase.
  - 1337 — In Greek, there are several lowercase forms of some letters; the one to use depends on its  
 1338 position in the word. Arabic has similar rules.
- 1339 • Many East Asian languages, including Japanese, Chinese, and Korean, do not distinguish  
 1340 case and are sometimes encoded in character sets that use more than one byte per character.
- 1341 • Multiple character codes may be used on the same machine simultaneously. There are  
 1342 several ISO character sets for European alphabets. In Japan, several Japanese character codes  
 1343 are commonly used together, sometimes even in filenames; this is evidently also the case in  
 1344 China. To handle case insensitivity, the kernel would have to at least be able to distinguish  
 1345 for which character sets the concept made sense.
- 1346 • The file system implementation historically deals only with bytes, not with characters, except  
 1347 for slash and the null byte.
- 1348 • The purpose of POSIX.1 is to standardize the common, existing definition, not to change it.  
 1349 Mandating case-insensitivity would make all historical implementations non-standard.

- Not only the interface, but also application programs would need to change, counter to the purpose of having minimal changes to existing application code.
- At least one of the original developers of the UNIX system has expressed objection in the strongest terms to either requiring case-insensitivity or making it an option, mostly on the basis that POSIX.1 should not hinder portability of application programs across related implementations in order to allow compatibility with unrelated operating systems.

Two proposals were entertained regarding case folding in filenames:

1. Remove all wording that previously permitted case folding.

Rationale Case folding is inconsistent with portable filename character set definition and filename definition (all characters except slash and null). No known implementations allowing all characters except slash and null also do case folding.

2. Change “though this practice is not recommended:” to “although this practice is strongly discouraged.”

Rationale If case folding must be included in POSIX.1, the wording should be stronger to discourage the practice.

The consensus selected the first proposal. Otherwise, a conforming application would have to assume that case folding would occur when it was not wanted, but that it would not occur when it was wanted.

#### A.4.7 File Times Update

This section reflects the actions of historical implementations. The times are not updated immediately, but are only marked for update by the functions. An implementation may update these times immediately.

The accuracy of the time update values is intentionally left unspecified so that systems can control the bandwidth of a possible covert channel.

The wording was carefully chosen to make it clear that there is no requirement that the conformance document contain information that might incidentally affect file update times. Any function that performs pathname resolution might update several *st\_atime* fields. Functions such as *getpwnam()* and *getgrnam()* might update the *st\_atime* field of some specific file or files. It is intended that these are not required to be documented in the conformance document, but they should appear in the system documentation.

#### A.4.8 Host and Network Byte Order

There is no additional rationale provided for this section.

#### A.4.9 Measurement of Execution Time

The methods used to measure the execution time of processes and threads, and the precision of these measurements, may vary considerably depending on the software architecture of the implementation, and on the underlying hardware. Implementations can also make tradeoffs between the scheduling overhead and the precision of the execution time measurements. IEEE Std 1003.1-2001 does not impose any requirement on the accuracy of the execution time; it instead specifies that the measurement mechanism and its precision are implementation-defined.

#### 1391 **A.4.10 Memory Synchronization**

1392 In older multi-processors, access to memory by the processors was strictly multiplexed. This  
 1393 meant that a processor executing program code interrogates or modifies memory in the order  
 1394 specified by the code and that all the memory operation of all the processors in the system  
 1395 appear to happen in some global order, though the operation histories of different processors are  
 1396 interleaved arbitrarily. The memory operations of such machines are said to be sequentially  
 1397 consistent. In this environment, threads can synchronize using ordinary memory operations. For  
 1398 example, a producer thread and a consumer thread can synchronize access to a circular data  
 1399 buffer as follows:

```

1400 int rdptr = 0;
1401 int wrptr = 0;
1402 data_t buf[BUFSIZE];

1403 Thread 1:
1404 while (work_to_do) {
1405 int next;

1406 buf[wrptr] = produce();
1407 next = (wrptr + 1) % BUFSIZE;
1408 while (rdptr == next)
1409 ;
1410 wrptr = next;
1411 }

1412 Thread 2:
1413 while (work_to_do) {
1414 while (rdptr == wrptr)
1415 ;
1416 consume(buf[rdptr]);
1417 rdptr = (rdptr + 1) % BUFSIZE;
1418 }

```

1419 In modern multi-processors, these conditions are relaxed to achieve greater performance. If one  
 1420 processor stores values in location A and then location B, then other processors loading data  
 1421 from location B and then location A may see the new value of B but the old value of A. The  
 1422 memory operations of such machines are said to be weakly ordered. On these machines, the  
 1423 circular buffer technique shown in the example will fail because the consumer may see the new  
 1424 value of *wrptr* but the old value of the data in the buffer. In such machines, synchronization can  
 1425 only be achieved through the use of special instructions that enforce an order on memory  
 1426 operations. Most high-level language compilers only generate ordinary memory operations to  
 1427 take advantage of the increased performance. They usually cannot determine when memory  
 1428 operation order is important and generate the special ordering instructions. Instead, they rely on  
 1429 the programmer to use synchronization primitives correctly to ensure that modifications to a  
 1430 location in memory are ordered with respect to modifications and/or access to the same location  
 1431 in other threads. Access to read-only data need not be synchronized. The resulting program is  
 1432 said to be data race-free.

1433 Synchronization is still important even when accessing a single primitive variable (for example,  
 1434 an integer). On machines where the integer may not be aligned to the bus data width or be larger  
 1435 than the data width, a single memory load may require multiple memory cycles. This means  
 1436 that it may be possible for some parts of the integer to have an old value while other parts have a  
 1437 newer value. On some processor architectures this cannot happen, but portable programs cannot  
 1438 rely on this.

In summary, a portable multi-threaded program, or a multi-process program that shares writable memory between processes, has to use the synchronization primitives to synchronize data access. It cannot rely on modifications to memory being observed by other threads in the order written in the application or even on modification of a single variable being seen atomically.

Conforming applications may only use the functions listed to synchronize threads of control with respect to memory access. There are many other candidates for functions that might also be used. Examples are: signal sending and reception, or pipe writing and reading. In general, any function that allows one thread of control to wait for an action caused by another thread of control is a candidate. IEEE Std 1003.1-2001 does not require these additional functions to synchronize memory access since this would imply the following:

- All these functions would have to be recognized by advanced compilation systems so that memory operations and calls to these functions are not reordered by optimization.
- All these functions would potentially have to have memory synchronization instructions added, depending on the particular machine.
- The additional functions complicate the model of how memory is synchronized and make automatic data race detection techniques impractical.

Formal definitions of the memory model were rejected as unreadable by the vast majority of programmers. In addition, most of the formal work in the literature has concentrated on the memory as provided by the hardware as opposed to the application programmer through the compiler and runtime system. It was believed that a simple statement intuitive to most programmers would be most effective. IEEE Std 1003.1-2001 defines functions that can be used to synchronize access to memory, but it leaves open exactly how one relates those functions to the semantics of each function as specified elsewhere in IEEE Std 1003.1-2001. IEEE Std 1003.1-2001 also does not make a formal specification of the partial ordering in time that the functions can impose, as that is implied in the description of the semantics of each function. It simply states that the programmer has to ensure that modifications do not occur “simultaneously” with other access to a memory location.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/4 is applied, adding a new paragraph beneath the table of functions: “The *pthread\_once()* function shall synchronize memory for the first call in each thread for a given **pthread\_once\_t** object.”.

#### A.4.11 Pathname Resolution

It is necessary to differentiate between the definition of pathname and the concept of pathname resolution with respect to the handling of trailing slashes. By specifying the behavior here, it is not possible to provide an implementation that is conforming but extends all interfaces that handle pathnames to also handle strings that are not legal pathnames (because they have trailing slashes).

Pathnames that end with one or more trailing slash characters must refer to directory paths. Previous versions of IEEE Std 1003.1-2001 were not specific about the distinction between trailing slashes on files and directories, and both were permitted.

Two types of implementation have been prevalent; those that ignored trailing slash characters on all pathnames regardless, and those that permitted them only on existing directories.

IEEE Std 1003.1-2001 requires that a pathname with a trailing slash character be treated as if it had a trailing “/ . ” everywhere.

Note that this change does not break any conforming applications; since there were two different types of implementation, no application could have portably depended on either

behavior. This change does however require some implementations to be altered to remain compliant. Substantial discussion over a three-year period has shown that the benefits to application developers outweighs the disadvantages for some vendors.

On a historical note, some early applications automatically appended a `'/'` to every path. Rather than fix the applications, the system implementation was modified to accept this behavior by ignoring any trailing slash.

Each directory has exactly one parent directory which is represented by the name **dot-dot** in the first directory. No other directory, regardless of linkages established by symbolic links, is considered the parent directory by IEEE Std 1003.1-2001.

There are two general categories of interfaces involving pathname resolution: those that follow the symbolic link, and those that do not. There are several exceptions to this rule; for example, `open(path, O_CREAT|O_EXCL)` will fail when `path` names a symbolic link. However, in all other situations, the `open()` function will follow the link.

What the filename **dot-dot** refers to relative to the root directory is implementation-defined. In Version 7 it refers to the root directory itself; this is the behavior mentioned in IEEE Std 1003.1-2001. In some networked systems the construction `../hostname/` is used to refer to the root directory of another host, and POSIX.1 permits this behavior.

Other networked systems use the construct `//hostname` for the same purpose; that is, a double initial slash is used. There is a potential problem with existing applications that create full pathnames by taking a trunk and a relative pathname and making them into a single string separated by `'/'`, because they can accidentally create networked pathnames when the trunk is `'/'`. This practice is not prohibited because such applications can be made to conform by simply changing to use `"/"` as a separator instead of `'/'`:

- If the trunk is `'/'`, the full pathname will begin with `"/"` (the initial `'/'` and the separator `"/"`). This is the same as `'/'`, which is what is desired. (This is the general case of making a relative pathname into an absolute one by prefixing with `"/"` instead of `'/'`.)
- If the trunk is `"/A"`, the result is `"/A/..."`; since non-leading sequences of two or more slashes are treated as a single slash, this is equivalent to the desired `"/A/..."`.
- If the trunk is `"//A"`, the implementation-defined semantics will apply. (The multiple slash rule would apply.)

Application developers should avoid generating pathnames that start with `"/"`. Implementations are strongly encouraged to avoid using this special interpretation since a number of applications currently do not follow this practice and may inadvertently generate `"//..."`.

The term “root directory” is only defined in POSIX.1 relative to the process. In some implementations, there may be no absolute root directory. The initialization of the root directory of a process is implementation-defined.

1522 **A.4.12 Process ID Reuse**

1523 There is no additional rationale provided for this section.

1524 **A.4.13 Scheduling Policy**

1525 There is no additional rationale provided for this section.

1526 **A.4.14 Seconds Since the Epoch**

1527 Coordinated Universal Time (UTC) includes leap seconds. However, in POSIX time (seconds  
 1528 since the Epoch), leap seconds are ignored (not applied) to provide an easy and compatible  
 1529 method of computing time differences. Broken-down POSIX time is therefore not necessarily  
 1530 UTC, despite its appearance.

1531 As of September 2000, 24 leap seconds had been added to UTC since the Epoch, 1 January, 1970.  
 1532 Historically, one leap second is added every 15 months on average, so this offset can be expected  
 1533 to grow steadily with time.

1534 Most systems' notion of "time" is that of a continuously increasing value, so this value should  
 1535 increase even during leap seconds. However, not only do most systems not keep track of leap  
 1536 seconds, but most systems are probably not synchronized to any standard time reference.  
 1537 Therefore, it is inappropriate to require that a time represented as seconds since the Epoch  
 1538 precisely represent the number of seconds between the referenced time and the Epoch.

1539 It is sufficient to require that applications be allowed to treat this time as if it represented the  
 1540 number of seconds between the referenced time and the Epoch. It is the responsibility of the  
 1541 vendor of the system, and the administrator of the system, to ensure that this value represents  
 1542 the number of seconds between the referenced time and the Epoch as closely as necessary for the  
 1543 application being run on that system.

1544 It is important that the interpretation of time names and seconds since the Epoch values be  
 1545 consistent across conforming systems; that is, it is important that all conforming systems  
 1546 interpret "536 457 599 seconds since the Epoch" as 59 seconds, 59 minutes, 23 hours 31 December  
 1547 1986, regardless of the accuracy of the system's idea of the current time. The expression is given  
 1548 to ensure a consistent interpretation, not to attempt to specify the calendar. The relationship  
 1549 between *tm\_yday* and the day of week, day of month, and month is in accordance with the  
 1550 Gregorian calendar, and so is not specified in POSIX.1.

1551 Consistent interpretation of seconds since the Epoch can be critical to certain types of distributed  
 1552 applications that rely on such timestamps to synchronize events. The accrual of leap seconds in  
 1553 a time standard is not predictable. The number of leap seconds since the Epoch will likely  
 1554 increase. POSIX.1 is more concerned about the synchronization of time between applications of  
 1555 astronomically short duration.

1556 Note that *tm\_yday* is zero-based, not one-based, so the day number in the example above is 364.  
 1557 Note also that the division is an integer division (discarding remainder) as in the C language.

1558 Note also that the meaning of *gmtime()*, *localtime()*, and *mktime()* is specified in terms of this  
 1559 expression. However, the ISO C standard computes *tm\_yday* from *tm\_mday*, *tm\_mon*, and  
 1560 *tm\_year* in *mktime()*. Because it is stated as a (bidirectional) relationship, not a function, and  
 1561 because the conversion between month-day-year and day-of-year dates is presumed well known  
 1562 and is also a relationship, this is not a problem.

1563 Implementations that implement **time\_t** as a signed 32-bit integer will overflow in 2038. The  
 1564 data size for **time\_t** is as per the ISO C standard definition, which is implementation-defined.

1565 See also **Epoch** (on page 17).

1566 The topic of whether seconds since the Epoch should account for leap seconds has been debated  
1567 on a number of occasions, and each time consensus was reached (with acknowledged dissent  
1568 each time) that the majority of users are best served by treating all days identically. (That is, the  
1569 majority of applications were judged to assume a single length—as measured in seconds since  
1570 the Epoch—for all days. Thus, leap seconds are not applied to seconds since the Epoch.) Those  
1571 applications which do care about leap seconds can determine how to handle them in whatever  
1572 way those applications feel is best. This was particularly emphasized because there was  
1573 disagreement about what the best way of handling leap seconds might be. It is a practical  
1574 impossibility to mandate that a conforming implementation must have a fixed relationship to  
1575 any particular official clock (consider isolated systems, or systems performing “reruns” by  
1576 setting the clock to some arbitrary time).

1577 Note that as a practical consequence of this, the length of a second as measured by some external  
1578 standard is not specified. This unspecified second is nominally equal to an International System  
1579 (SI) second in duration. Applications must be matched to a system that provides the particular  
1580 handling of external time in the way required by the application.

#### 1581 **A.4.15 Semaphore**

1582 There is no additional rationale provided for this section.

#### 1583 **A.4.16 Thread-Safety**

1584 Where the interface of a function required by IEEE Std 1003.1-2001 precludes thread-safety, an  
1585 alternate thread-safe form is provided. The names of these thread-safe forms are the same as the  
1586 non-thread-safe forms with the addition of the suffix “\_r”. The suffix “\_r” is historical, where  
1587 the ‘r’ stood for “reentrant”.

1588 In some cases, thread-safety is provided by restricting the arguments to an existing function.

1589 See also Section B.2.9.1 (on page 163).

#### 1590 **A.4.17 Tracing**

1591 Refer to Section B.2.11 (on page 179).

#### 1592 **A.4.18 Treatment of Error Conditions for Mathematical Functions**

1593 There is no additional rationale provided for this section.

#### 1594 **A.4.19 Treatment of NaN Arguments for Mathematical Functions**

1595 There is no additional rationale provided for this section.

#### 1596 **A.4.20 Utility**

1597 There is no additional rationale provided for this section.



1598 **A.4.21 Variable Assignment**

1599 There is no additional rationale provided for this section.

1600 **A.5 File Format Notation**

1601 The notation for spaces allows some flexibility for application output. Note that an empty  
 1602 character position in *format* represents one or more <blank>s on the output (not *white space*,  
 1603 which can include <newline>s). Therefore, another utility that reads that output as its input  
 1604 must be prepared to parse the data using *scanf()*, *awk*, and so on. The 'Δ' character is used when  
 1605 exactly one <space> is output.

1606 The treatment of integers and spaces is different from the *printf()* function in that they can be  
 1607 surrounded with <blank>s. This was done so that, given a format such as:

1608 `"%d\n", <foo>`1609 the implementation could use a *printf()* call such as:1610 `printf("%6d\n", foo);`1611 and still conform. This notation is thus somewhat like *scanf()* in addition to *printf()*.

1612 The *printf()* function was chosen as a model because most of the standard developers were  
 1613 familiar with it. One difference from the C function *printf()* is that the l and h conversion  
 1614 specifier characters are not used. As expressed by the Shell and Utilities volume of  
 1615 IEEE Std 1003.1-2001, there is no differentiation between decimal values for type **int**, type **long**,  
 1616 or type **short**. The conversion specifications %d or %i should be interpreted as an arbitrary  
 1617 length sequence of digits. Also, no distinction is made between single precision and double  
 1618 precision numbers (**float** or **double** in C). These are simply referred to as floating-point numbers.

1619 Many of the output descriptions in the Shell and Utilities volume of IEEE Std 1003.1-2001 use the  
 1620 term “line”, such as:

1621 `"%s", <input line>`

1622 Since the definition of *line* includes the trailing <newline> already, there is no need to include a  
 1623 '\n' in the format; a double <newline> would otherwise result.

1624 **A.6 Character Set**1625 **A.6.1 Portable Character Set**

1626 The portable character set is listed in full so there is no dependency on the ISO/IEC 646:1991  
 1627 standard (or historically ASCII) encoded character set, although the set is identical to the  
 1628 characters defined in the International Reference version of the ISO/IEC 646:1991 standard.

1629 IEEE Std 1003.1-2001 poses no requirement that multiple character sets or codesets be supported,  
 1630 leaving this as a marketing differentiation for implementors. Although multiple charmap files  
 1631 are supported, it is the responsibility of the implementation to provide the file(s); if only one is  
 1632 provided, only that one will be accessible using the *localedef -f* option.

1633 The statement about invariance in codesets for the portable character set is worded to avoid  
 1634 precluding implementations where multiple incompatible codesets are available (for instance,  
 1635 ASCII and EBCDIC). The standard utilities cannot be expected to produce predictable results if  
 1636 they access portable characters that vary on the same implementation.

Not all character sets need include the portable character set, but each locale must include it. For example, a Japanese-based locale might be supported by a mixture of character sets: JIS X 0201 Roman (a Japanese version of the ISO/IEC 646:1991 standard), JIS X 0208, and JIS X 0201 Katakana. Not all of these character sets include the portable characters, but at least one does (JIS X 0201 Roman).

## A.6.2 Character Encoding

Encoding mechanisms based on single shifts, such as the EUC encoding used in some Asian and other countries, can be supported via the current charmap mechanism. With single-shift encoding, each character is preceded by a shift code (SS2 or SS3). A complete EUC code, consisting of the portable character set (G0) and up to three additional character sets (G1, G2, G3), can be described using the current charmap mechanism; the encoding for each character in additional character sets G2 and G3 must then include their single-shift code. Other mechanisms to support locales based on encoding mechanisms such as locking shift are not addressed by this volume of IEEE Std 1003.1-2001.

## A.6.3 C Language Wide-Character Codes

There is no additional rationale provided for this section.

## A.6.4 Character Set Description File

IEEE PASC Interpretation 1003.2 #196 is applied, removing three lines of text dealing with ranges of symbolic names using position constant values which had been erroneously included in the final IEEE P1003.2b draft standard.

### A.6.4.1 State-Dependent Character Encodings

A requirement was considered that would force utilities to eliminate any redundant locking shifts, but this was left as a quality of implementation issue.

This change satisfies the following requirement from the ISO POSIX-2:1993 standard, Annex H.1:

*The support of state-dependent (shift encoding) character sets should be addressed fully. See descriptions of these in the Base Definitions volume of IEEE Std 1003.1-2001, Section 6.2, Character Encoding. If such character encodings are supported, it is expected that this will impact the Base Definitions volume of IEEE Std 1003.1-2001, Section 6.2, Character Encoding, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular Expressions, and the comm, cut, diff, grep, head, join, paste, and tail utilities.*

The character set description file provides:

- The capability to describe character set attributes (such as collation order or character classes) independent of character set encoding, and using only the characters in the portable character set. This makes it possible to create generic *localedef* source files for all codesets that share the portable character set (such as the ISO 8859 family or IBM Extended ASCII).
- Standardized symbolic names for all characters in the portable character set, making it possible to refer to any such character regardless of encoding.

Implementations are free to choose their own symbolic names, as long as the names identified by the Base Definitions volume of IEEE Std 1003.1-2001 are also defined; this provides support for already existing “character names”.

1679 The names selected for the members of the portable character set follow the  
 1680 ISO/IEC 8859-1:1998 standard and the ISO/IEC 10646-1:2000 standard. However, several  
 1681 commonly used UNIX system names occur as synonyms in the list:

- 1682 • The historical UNIX system names are used for control characters.
- 1683 • The word “slash” is given in addition to “solidus”.
- 1684 • The word “backslash” is given in addition to “reverse-solidus”.
- 1685 • The word “hyphen” is given in addition to “hyphen-minus”.
- 1686 • The word “period” is given in addition to “full-stop”.
- 1687 • For digits, the word “digit” is eliminated.
- 1688 • For letters, the words “Latin Capital Letter” and “Latin Small Letter” are eliminated.
- 1689 • The words “left brace” and “right brace” are given in addition to “left-curly-bracket” and  
 1690 “right-curly-bracket”.
- 1691 • The names of the digits are preferred over the numbers to avoid possible confusion between  
 1692 ‘0’ and ‘O’, and between ‘1’ and ‘l’ (one and the letter ell).

1693 The names for the control characters in the Base Definitions volume of IEEE Std 1003.1-2001,  
 1694 Chapter 6, Character Set were taken from the ISO/IEC 4873:1991 standard.

1695 The charmap file was introduced to resolve problems with the portability of, especially, *localedef*  
 1696 sources. IEEE Std 1003.1-2001 assumes that the portable character set is constant across all  
 1697 locales, but does not prohibit implementations from supporting two incompatible codings, such  
 1698 as both ASCII and EBCDIC. Such dual-support implementations should have all charmaps and  
 1699 *localedef* sources encoded using one portable character set, in effect cross-compiling for the other  
 1700 environment. Naturally, charmaps (and *localedef* sources) are only portable without  
 1701 transformation between systems using the same encodings for the portable character set. They  
 1702 can, however, be transformed between two sets using only a subset of the actual characters (the  
 1703 portable character set). However, the particular coded character set used for an application or an  
 1704 implementation does not necessarily imply different characteristics or collation; on the contrary,  
 1705 these attributes should in many cases be identical, regardless of codeset. The charmap provides  
 1706 the capability to define a common locale definition for multiple codesets (the same *localedef*  
 1707 source can be used for codesets with different extended characters; the ability in the charmap to  
 1708 define empty names allows for characters missing in certain codesets).

1709 The **<escape\_char>** declaration was added at the request of the international community to ease  
 1710 the creation of portable charmap files on terminals not implementing the default backslash  
 1711 escape. The **<comment\_char>** declaration was added at the request of the international  
 1712 community to eliminate the potential confusion between the number sign and the pound sign.

1713 The octal number notation with no leading zero required was selected to match those of *awk* and  
 1714 *tr* and is consistent with that used by *localedef*. To avoid confusion between an octal constant  
 1715 and the back-references used in *localedef* source, the octal, hexadecimal, and decimal constants  
 1716 must contain at least two digits. As single-digit constants are relatively rare, this should not  
 1717 impose any significant hardship. Provision is made for more digits to account for systems in  
 1718 which the byte size is larger than 8 bits. For example, a Unicode (ISO/IEC 10646-1:2000  
 1719 standard) system that has defined 16-bit bytes may require six octal, four hexadecimal, and five  
 1720 decimal digits.

1721 The decimal notation is supported because some newer international standards define character  
 1722 values in decimal, rather than in the old column/row notation.

The charmap identifies the coded character sets supported by an implementation. At least one charmap must be provided, but no implementation is required to provide more than one. Likewise, implementations can allow users to generate new charmaps (for instance, for a new version of the ISO 8859 family of coded character sets), but does not have to do so. If users are allowed to create new charmaps, the system documentation describes the rules that apply (for instance, “only coded character sets that are supersets of the ISO/IEC 646:1991 standard IRV, no multi-byte characters”).

This addition of the **WIDTH** specification satisfies the following requirement from the ISO POSIX-2:1993 standard, Annex H.1:

- (9) *The definition of column position relies on the implementation’s knowledge of the integral width of the characters. The charmap or LC\_CTYPE locale definitions should be enhanced to allow application specification of these widths.*

The character “width” information was first considered for inclusion under *LC\_CTYPE* but was moved because it is more closely associated with the information in the charmap than information in the locale source (cultural conventions information). Concerns were raised that formalizing this type of information is moving the locale source definition from the codeset-independent entity that it was designed to be to a repository of codeset-specific information. A similar issue occurred with the *<code\_set\_name>*, *<mb\_cur\_max>*, and *<mb\_cur\_min>* information, which was resolved to reside in the charmap definition.

The width definition was added to the IEEE P1003.2b draft standard with the intent that the *wcswidth()* and/or *wcwidth()* functions (currently specified in the System Interfaces volume of IEEE Std 1003.1-2001) be the mechanism to retrieve the character width information.

## A.7 Locale

### A.7.1 General

The description of locales is based on work performed in the UniForum Technical Committee, Subcommittee on Internationalization. Wherever appropriate, keywords are taken from the ISO C standard or the X/Open Portability Guide.

The value used to specify a locale with environment variables is the name specified as the *name* operand to the *localedef* utility when the locale was created. This provides a verifiable method to create and invoke a locale.

The “object” definitions need not be portable, as long as “source” definitions are. Strictly speaking, source definitions are portable only between implementations using the same character set(s). Such source definitions, if they use symbolic names only, easily can be ported between systems using different codesets, as long as the characters in the portable character set (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set) have common values between the codesets; this is frequently the case in historical implementations. Of source, this requires that the symbolic names used for characters outside the portable character set be identical between character sets. The definition of symbolic names for characters is outside the scope of IEEE Std 1003.1-2001, but is certainly within the scope of other standards organizations.

Applications can select the desired locale by invoking the *setlocale()* function (or equivalent) with the appropriate value. If the function is invoked with an empty string, the value of the corresponding environment variable is used. If the environment variable is not set or is set to the empty string, the implementation sets the appropriate environment as defined in the Base

1767 Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

## 1768 **A.7.2 POSIX Locale**

1769 The POSIX locale is equal to the C locale. To avoid being classified as a C-language function, the  
1770 name has been changed to the POSIX locale; the environment variable value can be either  
1771 "POSIX" or, for historical reasons, "C".

1772 The POSIX definitions mirror the historical UNIX system behavior.

1773 The use of symbolic names for characters in the tables does not imply that the POSIX locale must  
1774 be described using symbolic character names, but merely that it may be advantageous to do so.

## 1775 **A.7.3 Locale Definition**

1776 The decision to separate the file format from the *localedef* utility description was only partially  
1777 editorial. Implementations may provide other interfaces than *localedef*. Requirements on “the  
1778 utility”, mostly concerning error messages, are described in this way because they are meant to  
1779 affect the other interfaces implementations may provide as well as *localedef*.

1780 The text about POSIX2\_LOCALEDEF does not mean that internationalization is optional; only  
1781 that the functionality of the *localedef* utility is. REs, for instance, must still be able to recognize,  
1782 for example, character class expressions such as "[[:alpha:]]". A possible analogy is with  
1783 an applications development environment; while all conforming implementations must be  
1784 capable of executing applications, not all need to have the development environment installed.  
1785 The assumption is that the capability to modify the behavior of utilities (and applications) via  
1786 locale settings must be supported. If the *localedef* utility is not present, then the only choice is to  
1787 select an existing (presumably implementation-documented) locale. An implementation could,  
1788 for example, choose to support only the POSIX locale, which would in effect limit the amount of  
1789 changes from historical implementations quite drastically. The *localedef* utility is still required,  
1790 but would always terminate with an exit code indicating that no locale could be created.  
1791 Supported locales must be documented using the syntax defined in this chapter. (This ensures  
1792 that users can accurately determine what capabilities are provided. If the implementation  
1793 decides to provide additional capabilities to the ones in this chapter, that is already provided  
1794 for.)

1795 If the option is present (that is, locales can be created), then the *localedef* utility must be capable  
1796 of creating locales based on the syntax and rules defined in this chapter. This does not mean that  
1797 the implementation cannot also provide alternate means for creating locales.

1798 The octal, decimal, and hexadecimal notations are the same employed by the charmap facility  
1799 (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 6.4, Character Set Description  
1800 File). To avoid confusion between an octal constant and a back-reference, the octal, hexadecimal,  
1801 and decimal constants must contain at least two digits. As single-digit constants are relatively  
1802 rare, this should not impose any significant hardship. Provision is made for more digits to  
1803 account for systems in which the byte size is larger than 8 bits. For example, a Unicode (see the  
1804 ISO/IEC 10646-1:2000 standard) system that has defined 16-bit bytes may require six octal, four  
1805 hexadecimal, and five decimal digits. As with the charmap file, multi-byte characters are  
1806 described in the locale definition file using “big-endian” notation for reasons of portability.  
1807 There is no requirement that the internal representation in the computer memory be in this same  
1808 order.

1809 One of the guidelines used for the development of this volume of IEEE Std 1003.1-2001 is that  
1810 characters outside the invariant part of the ISO/IEC 646:1991 standard should not be used in  
1811 portable specifications. The backslash character is not in the invariant part; the number sign is,  
1812 but with multiple representations: as a number sign, and as a pound sign. As far as general

usage of these symbols, they are covered by the “grandfather clause”, but for newly defined interfaces, the WG15 POSIX working group has requested that POSIX provide alternate representations. Consequently, while the default escape character remains the backslash and the default comment character is the number sign, implementations are required to recognize alternative representations, identified in the applicable source file via the `<escape_char>` and `<comment_char>` keywords.

#### A.7.3.1 *LC\_CTYPE*

The *LC\_CTYPE* category is primarily used to define the encoding-independent aspects of a character set, such as character classification. In addition, certain encoding-dependent characteristics are also defined for an application via the *LC\_CTYPE* category. IEEE Std 1003.1-2001 does not mandate that the encoding used in the locale is the same as the one used by the application because an implementation may decide that it is advantageous to define locales in a system-wide encoding rather than having multiple, logically identical locales in different encodings, and to convert from the application encoding to the system-wide encoding on usage. Other implementations could require encoding-dependent locales.

In either case, the *LC\_CTYPE* attributes that are directly dependent on the encoding, such as `<mb_cur_max>` and the display width of characters, are not user-specifiable in a locale source and are consequently not defined as keywords.

Implementations may define additional keywords or extend the *LC\_CTYPE* mechanism to allow application-defined keywords.

The text “The ellipsis specification shall only be valid within a single encoded character set” is present because it is possible to have a locale supported by multiple character encodings, as explained in the rationale for the Base Definitions volume of IEEE Std 1003.1-2001, Section 6.1, Portable Character Set. An example given there is of a possible Japanese-based locale supported by a mixture of the character sets JIS X 0201 Roman, JIS X 0208, and JIS X 0201 Katakana. Attempting to express a range of characters across these sets is not logical and the implementation is free to reject such attempts.

As the *LC\_CTYPE* character classes are based on the ISO C standard character class definition, the category does not support multi-character elements. For instance, the German character `<sharp-s>` is traditionally classified as a lowercase letter. There is no corresponding uppercase letter; in proper capitalization of German text, the `<sharp-s>` will be replaced by “SS”; that is, by two characters. This kind of conversion is outside the scope of the **toupper** and **tolower** keywords.

Where IEEE Std 1003.1-2001 specifies that only certain characters can be specified, as for the keywords **digit** and **xdigit**, the specified characters must be from the portable character set, as shown. As an example, only the Arabic digits 0 through 9 are acceptable as digits.

The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included characters. These only need to be specified if the character values (that is, encoding) differs from the implementation default values. It is not possible to define a locale without these automatically included characters unless some implementation extension is used to prevent their inclusion. Such a definition would not be a proper superset of the C locale, and thus, it might not be possible for the standard utilities to be implemented as programs conforming to the ISO C standard.

The definition of character class **digit** requires that only ten characters—the ones defining digits—can be specified; alternate digits (for example, Hindi or Kanji) cannot be specified here. However, the encoding may vary if an implementation supports more than one encoding.

The definition of character class **xdigit** requires that the characters included in character class **digit** are included here also and allows for different symbols for the hexadecimal digits 10 through 15.

The inclusion of the **charclass** keyword satisfies the following requirement from the ISO POSIX-2: 1993 standard, Annex H.1:

(3) *The LC\_CTYPE (2.5.2.1) locale definition should be enhanced to allow user-specified additional character classes, similar in concept to the ISO C standard Multibyte Support Extension (MSE) iswctype() function.*

This keyword was previously included in The Open Group specifications and is now mandated in the Shell and Utilities volume of IEEE Std 1003.1-2001.

The symbolic constant {CHARCLASS\_NAME\_MAX} was also adopted from The Open Group specifications. Applications portability is enhanced by the use of symbolic constants.

#### A.7.3.2 LC\_COLLATE

The rules governing collation depend to some extent on the use. At least five different levels of increasingly complex collation rules can be distinguished:

1. *Byte/machine code order:* This is the historical collation order in the UNIX system and many proprietary operating systems. Collation is here performed character by character, without any regard to context. The primary virtue is that it usually is quite fast and also completely deterministic; it works well when the native machine collation sequence matches the user expectations.
2. *Character order:* On this level, collation is also performed character by character, without regard to context. The order between characters is, however, not determined by the code values, but on the expectations by the user of the “correct” order between characters. In addition, such a (simple) collation order can specify that certain characters collate equally (for example, uppercase and lowercase letters).
3. *String ordering:* On this level, entire strings are compared based on relatively straightforward rules. Several “passes” may be required to determine the order between two strings. Characters may be ignored in some passes, but not in others; the strings may be compared in different directions; and simple string substitutions may be performed before strings are compared. This level is best described as “dictionary” ordering; it is based on the spelling, not the pronunciation, or meaning, of the words.
4. *Text search ordering:* This is a further refinement of the previous level, best described as “telephone book ordering”; some common homonyms (words spelled differently but with the same pronunciation) are collated together; numbers are collated as if they were spelled out, and so on.
5. *Semantic-level ordering:* Words and strings are collated based on their meaning; entire words (such as “the”) are eliminated; the ordering is not deterministic. This usually requires special software and is highly dependent on the intended use.

While the historical collation order formally is at level 1, for the English language it corresponds roughly to elements at level 2. The user expects to see the output from the *ls* utility sorted very much as it would be in a dictionary. While telephone book ordering would be an optimal goal for standard collation, this was ruled out as the order would be language-dependent. Furthermore, a requirement was that the order must be determined solely from the text string and the collation rules; no external information (for example, “pronunciation dictionaries”) could be required.

1904 As a result, the goal for the collation support is at level 3. This also matches the requirements for  
 1905 the Canadian collation order, as well as other, known collation requirements for alphabetic  
 1906 scripts. It specifically rules out collation based on pronunciation rules or based on semantic  
 1907 analysis of the text.

1908 The syntax for the *LC\_COLLATE* category source meets the requirements for level 3 and has  
 1909 been verified to produce the correct result with examples based on French, Canadian, and  
 1910 Danish collation order. Because it supports multi-character collating elements, it is also capable  
 1911 of supporting collation in codesets where a character is expressed using non-spacing characters  
 1912 followed by the base character (such as the ISO/IEC 6937: 1994 standard).

1913 The directives that can be specified in an operand to the **order\_start** keyword are based on the  
 1914 requirements specified in several proposed standards and in customary use. The following is a  
 1915 rephrasing of rules defined for “lexical ordering in English and French” by the Canadian  
 1916 Standards Association (the text in square brackets is rephrased):

- 1917 • Once special characters [punctuation] have been removed from original strings, the ordering  
 1918 is determined by scanning forwards (left to right) [disregarding case and diacriticals].
- 1919 • In case of equivalence, special characters are once again removed from original strings and  
 1920 the ordering is determined by scanning backwards (starting from the rightmost character of  
 1921 the string and back), character by character [disregarding case but considering diacriticals].
- 1922 • In case of repeated equivalence, special characters are removed again from original strings  
 1923 and the ordering is determined by scanning forwards, character by character [considering  
 1924 both case and diacriticals].
- 1925 • If there is still an ordering equivalence after the first three rules have been applied, then only  
 1926 special characters and the position they occupy in the string are considered to determine  
 1927 ordering. The string that has a special character in the lowest position comes first. If two  
 1928 strings have a special character in the same position, the character [with the lowest collation  
 1929 value] comes first. In case of equality, the other special characters are considered until there  
 1930 is a difference or until all special characters have been exhausted.

1931 It is estimated that this part of IEEE Std 1003.1-2001 covers the requirements for all European  
 1932 languages, and no particular problems are anticipated with Slavic or Middle East character sets.

1933 The Far East (particularly Japanese/Chinese) collations are often based on contextual  
 1934 information and pronunciation rules (the same ideogram can have different meanings and  
 1935 different pronunciations). Such collation, in general, falls outside the desired goal of  
 1936 IEEE Std 1003.1-2001. There are, however, several other collation rules (stroke/radical or “most  
 1937 common pronunciation”) that can be supported with the mechanism described here.

1938 The character order is defined by the order in which characters and elements are specified  
 1939 between the **order\_start** and **order\_end** keywords. Weights assigned to the characters and  
 1940 elements define the collation sequence; in the absence of weights, the character order is also the  
 1941 collation sequence.

1942 The **position** keyword provides the capability to consider, in a compare, the relative position of  
 1943 characters not subject to **IGNORE**. As an example, consider the two strings "o-ring" and  
 1944 "or-ing". Assuming the hyphen is subject to **IGNORE** on the first pass, the two strings  
 1945 compare equal, and the position of the hyphen is immaterial. On second pass, all characters  
 1946 except the hyphen are subject to **IGNORE**, and in the normal case the two strings would again  
 1947 compare equal. By taking position into account, the first collates before the second.



1948 A.7.3.3 *LC\_MONETARY*

1949 The currency symbol does not appear in *LC\_MONETARY* because it is not defined in the C locale  
 1950 of the ISO C standard.

1951 The ISO C standard limits the size of decimal points and thousands delimiters to single-byte  
 1952 values. In locales based on multi-byte coded character sets, this cannot be enforced;  
 1953 IEEE Std 1003.1-2001 does not prohibit such characters, but makes the behavior unspecified (in  
 1954 the text “In contexts where other standards ...”).

1955 The grouping specification is based on, but not identical to, the ISO C standard. The `-1` indicates  
 1956 that no further grouping is performed; the equivalent of `{CHAR_MAX}` in the ISO C standard.

1957 The text “the value is not available in the locale” is taken from the ISO C standard and is used  
 1958 instead of the “unspecified” text in early proposals. There is no implication that omitting these  
 1959 keywords or assigning them values of “ ” or `-1` produces unspecified results; such omissions or  
 1960 assignments eliminate the effects described for the keyword or produce zero-length strings, as  
 1961 appropriate.

1962 The locale definition is an extension of the ISO C standard *localeconv()* specification. In  
 1963 particular, rules on how **currency\_symbol** is treated are extended to also cover **int\_curr\_symbol**,  
 1964 and **p\_set\_by\_space** and **n\_sep\_by\_space** have been augmented with the value 2, which places  
 1965 a <space> between the sign and the symbol (if they are adjacent; otherwise, it should be treated  
 1966 as a 0). The following table shows the result of various combinations:

|                          |                        | <b>p_sep_by_space</b> |           |          |
|--------------------------|------------------------|-----------------------|-----------|----------|
|                          |                        | <b>2</b>              | <b>1</b>  | <b>0</b> |
| <b>p_cs_precedes = 1</b> | <b>p_sign_posn = 0</b> | (\$1.25)              | (\$ 1.25) | (\$1.25) |
|                          | <b>p_sign_posn = 1</b> | + \$1.25              | +\$ 1.25  | +\$1.25  |
|                          | <b>p_sign_posn = 2</b> | \$1.25 +              | \$ 1.25+  | \$1.25+  |
|                          | <b>p_sign_posn = 3</b> | + \$1.25              | +\$ 1.25  | +\$1.25  |
|                          | <b>p_sign_posn = 4</b> | \$ +1.25              | \$+ 1.25  | +\$1.25  |
| <b>p_cs_precedes = 0</b> | <b>p_sign_posn = 0</b> | (1.25 \$)             | (1.25 \$) | (1.25\$) |
|                          | <b>p_sign_posn = 1</b> | +1.25 \$              | +1.25 \$  | +1.25\$  |
|                          | <b>p_sign_posn = 2</b> | 1.25\$ +              | 1.25 \$+  | 1.25\$+  |
|                          | <b>p_sign_posn = 3</b> | 1.25+ \$              | 1.25 +\$  | 1.25+\$  |
|                          | <b>p_sign_posn = 4</b> | 1.25\$ +              | 1.25 \$+  | 1.25\$+  |

1979 The following is an example of the interpretation of the **mon\_grouping** keyword. Assuming that  
 1980 the value to be formatted is 123456789 and the **mon\_thousands\_sep** is ‘ ’, then the following  
 1981 table shows the result. The third column shows the equivalent string in the ISO C standard that  
 1982 would be used by the *localeconv()* function to accommodate this grouping.

| <b>mon_grouping</b> | <b>Formatted Value</b> | <b>ISO C String</b> |
|---------------------|------------------------|---------------------|
| 3;-1                | 123456'789             | "\3\177"            |
| 3                   | 123'456'789            | "\3"                |
| 3;2;-1              | 1234'56'789            | "\3\2\177"          |
| 3;2                 | 12'34'56'789           | "\3\2"              |
| -1                  | 123456789              | "\177"              |

1989 In these examples, the octal value of `{CHAR_MAX}` is 177.

1990 IEEE Std 1003.1-2001 / Cor 1-2002, item XBD / TC1 / D6 / 6 adds a correction that permits the Euro  
 1991 currency symbol and addresses extensibility. The correction is stated using the term “should”  
 1992 intentionally, in order to make this a recommendation rather than a restriction on  
 1993 implementations. This allows for flexibility in implementations on how they handle future

|      |                                                                                                                                   |  |
|------|-----------------------------------------------------------------------------------------------------------------------------------|--|
| 1994 | currency symbol additions.                                                                                                        |  |
| 1995 | IEEE Std 1003.1-2001 / Cor 1-2002, tem XBD/TC1/D6/5 is applied, adding the <b>int[np]*</b> values                                 |  |
| 1996 | to the POSIX locale definition of <i>LC_MONETARY</i> .                                                                            |  |
| 1997 | <b>A.7.3.4</b> <i>LC_NUMERIC</i>                                                                                                  |  |
| 1998 | See the rationale for <i>LC_MONETARY</i> for a description of the behavior of grouping.                                           |  |
| 1999 | <b>A.7.3.5</b> <i>LC_TIME</i>                                                                                                     |  |
| 2000 | Although certain of the conversion specifications in the POSIX locale (such as the name of the                                    |  |
| 2001 | month) are shown with initial capital letters, this need not be the case in other locales. Programs                               |  |
| 2002 | using these conversion specifications may need to adjust the capitalization if the output is going                                |  |
| 2003 | to be used at the beginning of a sentence.                                                                                        |  |
| 2004 | The <i>LC_TIME</i> descriptions of <b>abday</b> , <b>day</b> , <b>mon</b> , and <b>abmon</b> imply a Gregorian style calendar (7- |  |
| 2005 | day weeks, 12-month years, leap years, and so on). Formatting time strings for other types of                                     |  |
| 2006 | calendars is outside the scope of IEEE Std 1003.1-2001.                                                                           |  |
| 2007 | While the ISO 8601:2000 standard numbers the weekdays starting with Monday, historical                                            |  |
| 2008 | practice is to use the Sunday as the first day. Rather than change the order and introduce                                        |  |
| 2009 | potential confusion, the days must be specified beginning with Sunday; previous references to                                     |  |
| 2010 | “first day” have been removed. Note also that the Shell and Utilities volume of                                                   |  |
| 2011 | IEEE Std 1003.1-2001 <i>date</i> utility supports numbering compliant with the ISO 8601:2000                                      |  |
| 2012 | standard.                                                                                                                         |  |
| 2013 | As specified under <i>date</i> in the Shell and Utilities volume of IEEE Std 1003.1-2001 and <i>strftime()</i> in                 |  |
| 2014 | the System Interfaces volume of IEEE Std 1003.1-2001, the conversion specifications                                               |  |
| 2015 | corresponding to the optional keywords consist of a modifier followed by a traditional                                            |  |
| 2016 | conversion specification (for instance, %Ex). If the optional keywords are not supported by the                                   |  |
| 2017 | implementation or are unspecified for the current locale, these modified conversion                                               |  |
| 2018 | specifications are treated as the traditional conversion specifications. For example, assume the                                  |  |
| 2019 | following keywords:                                                                                                               |  |
| 2020 | <code>alt_digits</code> "0th"; "1st"; "2nd"; "3rd"; "4th"; "5th"; \                                                               |  |
| 2021 | "6th"; "7th"; "8th"; "9th"; "10th"                                                                                                |  |
| 2022 | <code>d_fmt</code> "The %Od day of %B in %Y"                                                                                      |  |
| 2023 | On July 4th 1776, the %x conversion specifications would result in "The 4th day of July                                           |  |
| 2024 | in 1776", while on July 14th 1789 it would result in "The 14 day of July in 1789". It                                             |  |
| 2025 | can be noted that the above example is for illustrative purposes only; the %O modifier is                                         |  |
| 2026 | primarily intended to provide for Kanji or Hindi digits in <i>date</i> formats.                                                   |  |
| 2027 | The following is an example for Japan that supports the current plus last three Emperors and                                      |  |
| 2028 | reverts to Western style numbering for years prior to the Meiji era. The example also allows for                                  |  |
| 2029 | the custom of using a special name for the first year of an era instead of using 1. (The examples                                 |  |
| 2030 | substitute romaji where kanji should be used.)                                                                                    |  |

2031 era\_d\_fmt "%EY%mgatsu%dnichi (%a)"

2032 era "+:2:1990/01/01:+\*:Heisei:%EC%Eynen";\  
 2033 "+:1:1989/01/08:1989/12/31:Heisei:%ECgannen";\  
 2034 "+:2:1927/01/01:1989/01/07:Shouwa:%EC%Eynen";\  
 2035 "+:1:1926/12/25:1926/12/31:Shouwa:%ECgannen";\  
 2036 "+:2:1913/01/01:1926/12/24:Taishou:%EC%Eynen";\  
 2037 "+:1:1912/07/30:1912/12/31:Taishou:%ECgannen";\  
 2038 "+:2:1869/01/01:1912/07/29:Meiji:%EC%Eynen";\  
 2039 "+:1:1868/09/08:1868/12/31:Meiji:%ECgannen";\  
 2040 "-:1868:1868/09/07:-\*::%Ey"

2041 Assuming that the current date is September 21, 1991, a request to *date* or *strftime()* would yield  
 2042 the following results:

2043 %Ec - Heisei3nen9gatsu21nichi (Sat) 14:39:26  
 2044 %EC - Heisei  
 2045 %Ex - Heisei3nen9gatsu21nichi (Sat)  
 2046 %Ey - 3  
 2047 %EY - Heisei3nen

2048 Example era definitions for the Republic of China:

2049 era "+:2:1913/01/01:+\*:ChungHwaMingGuo:%EC%EyNen";\  
 2050 "+:1:1912/1/1:1912/12/31:ChungHwaMingGuo:%ECYuenNen";\  
 2051 "+:1:1911/12/31:-\*:MingChien:%EC%EyNen"

2052 Example definitions for the Christian Era:

2053 era "+:1:0001/01/01:+\*:AD:%EC %Ey";\  
 2054 "+:1:-0001/12/31:-\*:BC:%Ey %EC"

#### 2055 A.7.3.6 LC\_MESSAGES

2056 The **yesstr** and **nostr** locale keywords and the YESSTR and NOSTR *langinfo* items were formerly  
 2057 used to match user affirmative and negative responses. In IEEE Std 1003.1-2001, the **yesexpr**,  
 2058 **noexpr**, YESEXPR, and NOEXPR extended regular expressions have replaced them.  
 2059 Applications should use the general locale-based messaging facilities to issue prompting  
 2060 messages which include sample desired responses.

### 2061 A.7.4 Locale Definition Grammar

2062 There is no additional rationale provided for this section.

#### 2063 A.7.4.1 Locale Lexical Conventions

2064 There is no additional rationale provided for this section.

#### 2065 A.7.4.2 Locale Grammar

2066 There is no additional rationale provided for this section.

## 2067 A.7.5 Locale Definition Example

2068 The following is an example of a locale definition file that could be used as input to the *localedef*  
 2069 utility. It assumes that the utility is executed with the **-f** option, naming a charmap file with (at  
 2070 least) the following content:

```

2071 CHARMAP
2072 <space> \x20
2073 <dollar> \x24
2074 <A> \101
2075 <a> \141
2076 <A-acute> \346
2077 <a-acute> \365
2078 <A-grave> \300
2079 <a-grave> \366
2080 \142
2081 <C> \103
2082 <c> \143
2083 <c-cedilla> \347
2084 <d> \x64
2085 <H> \110
2086 <h> \150
2087 <eszet> \xb7
2088 <s> \x73
2089 <z> \x7a
2090 END CHARMAP

```

2091 It should not be taken as complete or to represent any actual locale, but only to illustrate the  
 2092 syntax.

```

2093 #
2094 LC_CTYPE
2095 lower <a>;;<c>;<c-cedilla>;<d>;...;<z>
2096 upper A;B;C;Ç;...;Z
2097 space \x20;\x09;\x0a;\x0b;\x0c;\x0d
2098 blank \040;\011
2099 toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
2100 END LC_CTYPE
2101 #
2102 LC_COLLATE
2103 #
2104 # The following example of collation is based on
2105 # Canadian standard Z243.4.1-1998, "Canadian Alphanumeric
2106 # Ordering Standard for Character Sets of CSA Z234.4 Standard".
2107 # (Other parts of this example locale definition file do not
2108 # purport to relate to Canada, or to any other real culture.)
2109 # The proposed standard defines a 4-weight collation, such that
2110 # in the first pass, characters are compared without regard to
2111 # case or accents; in the second pass, backwards-compare without
2112 # regard to case; in the third pass, forwards-compare without
2113 # regard to diacriticals. In the 3 first passes, non-alphabetic
2114 # characters are ignored; in the fourth pass, only special
2115 # characters are considered, such that "The string that has a
2116 # special character in the lowest position comes first. If two

```

```

2117 # strings have a special character in the same position, the
2118 # collation value of the special character determines ordering.
2119 #
2120 # Only a subset of the character set is used here; mostly to
2121 # illustrate the set-up.
2122 #
2123 collating-symbol <NULL>
2124 collating-symbol <LOW_VALUE>
2125 collating-symbol <LOWER-CASE>
2126 collating-symbol <SUBSCRIPT-LOWER>
2127 collating-symbol <SUPERSCRIPT-LOWER>
2128 collating-symbol <UPPER-CASE>
2129 collating-symbol <NO-ACCENT>
2130 collating-symbol <PECULIAR>
2131 collating-symbol <LIGATURE>
2132 collating-symbol <ACUTE>
2133 collating-symbol <GRAVE>
2134 # Further collating-symbols follow.
2135 #
2136 # Properly, the standard does not include any multi-character
2137 # collating elements; the one below is added for completeness.
2138 #
2139 collating_element <ch> from "<c><h>"
2140 collating_element <CH> from "<C><H>"
2141 collating_element <Ch> from "<C><h>"
2142 #
2143 order_start forward;backward;forward;forward,position
2144 #
2145 # Collating symbols are specified first in the sequence to allocate
2146 # basic collation values to them, lower than that of any character.
2147 <NULL>
2148 <LOW_VALUE>
2149 <LOWER-CASE>
2150 <SUBSCRIPT-LOWER>
2151 <SUPERSCRIPT-LOWER>
2152 <UPPER-CASE>
2153 <NO-ACCENT>
2154 <PECULIAR>
2155 <LIGATURE>
2156 <ACUTE>
2157 <GRAVE>
2158 <RING-ABOVE>
2159 <DIAERESIS>
2160 <TILDE>
2161 # Further collating symbols are given a basic collating value here.
2162 #
2163 # Here follow special characters.
2164 <space> IGNORE;IGNORE;IGNORE;<space>
2165 # Other special characters follow here.
2166 #
2167 # Here follow the regular characters.
2168 <a> <a>;<NO-ACCENT>;<LOWER-CASE>;IGNORE

```

```

2169 <A> <a>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
2170 <a-acute> <a>; <ACUTE>; <LOWER-CASE>; IGNORE
2171 <A-acute> <a>; <ACUTE>; <UPPER-CASE>; IGNORE
2172 <a-grave> <a>; <GRAVE>; <LOWER-CASE>; IGNORE
2173 <A-grave> <a>; <GRAVE>; <UPPER-CASE>; IGNORE
2174 <ae> "<a><e>"; "<LIGATURE><LIGATURE>"; \
2175 "<LOWER-CASE><LOWER-CASE>"; IGNORE
2176 <AE> "<a><e>"; "<LIGATURE><LIGATURE>"; \
2177 "<UPPER-CASE><UPPER-CASE>"; IGNORE
2178 ; <NO-ACCENT>; <LOWER-CASE>; IGNORE
2179 ; <NO-ACCENT>; <UPPER-CASE>; IGNORE
2180 <c> <c>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
2181 <C> <c>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
2182 <ch> <ch>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
2183 <Ch> <ch>; <NO-ACCENT>; <PECULIAR>; IGNORE
2184 <CH> <ch>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
2185 #
2186 # As an example, the strings "Bach" and "bach" could be encoded (for
2187 # compare purposes) as:
2188 # "Bach" ; <a>; <ch>; <LOW_VALUE>; <NO_ACCENT>; \
2189 # <NO_ACCENT>; <LOW_VALUE>; <UPPER-CASE>; <LOWER-CASE>; \
2190 # <LOWER-CASE>; <NULL>
2191 # "bach" ; <a>; <ch>; <LOW_VALUE>; <NO_ACCENT>; \
2192 # <NO_ACCENT>; <LOW_VALUE>; <LOWER-CASE>; <LOWER-CASE>; \
2193 # <LOWER-CASE>; <NULL>
2194 #
2195 # The two strings are equal in pass 1 and 2, but differ in pass 3.
2196 #
2197 # Further characters follow.
2198 #
2199 UNDEFINED IGNORE; IGNORE; IGNORE; IGNORE
2200 #
2201 order_end
2202 #
2203 END LC_COLLATE
2204 #
2205 LC_MONETARY
2206 int_curr_symbol "USD "
2207 currency_symbol "$"
2208 mon_decimal_point "."
2209 mon_grouping 3;0
2210 positive_sign ""
2211 negative_sign "- "
2212 p_cs_precedes 1
2213 n_sign_posn 0
2214 END LC_MONETARY
2215 #
2216 LC_NUMERIC
2217 copy "US_en.ASCII"
2218 END LC_NUMERIC
2219 #
2220 LC_TIME

```

```

2221 abday "Sun"; "Mon"; "Tue"; "Wed"; "Thu"; "Fri"; "Sat"
2222 #
2223 day "Sunday"; "Monday"; "Tuesday"; "Wednesday"; \
2224 "Thursday"; "Friday"; "Saturday"
2225 #
2226 abmon "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun"; \
2227 "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec"
2228 #
2229 mon "January"; "February"; "March"; "April"; \
2230 "May"; "June"; "July"; "August"; "September"; \
2231 "October"; "November"; "December"
2232 #
2233 d_t_fmt "%a %b %d %T %Z %Y\n"
2234 END LC_TIME
2235 #
2236 LC_MESSAGES
2237 yesexpr "^[yY] [[:alpha:]]*" | (OK) "
2238 #
2239 noexpr "^[nN] [[:alpha:]]*"
2240 END LC_MESSAGES

```

## 2241 **A.8 Environment Variables**

### 2242 **A.8.1 Environment Variable Definition**

2243 The variable *environ* is not intended to be declared in any header, but rather to be declared by the  
 2244 user for accessing the array of strings that is the environment. This is the traditional usage of the  
 2245 symbol. Putting it into a header could break some programs that use the symbol for their own  
 2246 purposes.

2247 The decision to restrict conforming systems to the use of digits, uppercase letters, and  
 2248 underscores for environment variable names allows applications to use lowercase letters in their  
 2249 environment variable names without conflicting with any conforming system.

2250 In addition to the obvious conflict with the shell syntax for positional parameter substitution,  
 2251 some historical applications (including some shells) exclude names with leading digits from the  
 2252 environment.

### 2253 **A.8.2 Internationalization Variables**

2254 The text about locale implies that any utilities written in standard C and conforming to  
 2255 IEEE Std 1003.1-2001 must issue the following call:

```
2256 setlocale(LC_ALL, "")
```

2257 If this were omitted, the ISO C standard specifies that the C locale would be used.

2258 If any of the environment variables are invalid, it makes sense to default to an implementation-  
 2259 defined, consistent locale environment. It is more confusing for a user to have partial settings  
 2260 occur in case of a mistake. All utilities would then behave in one language/cultural  
 2261 environment. Furthermore, it provides a way of forcing the whole environment to be the  
 2262 implementation-defined default. Disastrous results could occur if a pipeline of utilities partially  
 2263 uses the environment variables in different ways. In this case, it would be appropriate for  
 2264 utilities that use *LANG* and related variables to exit with an error if any of the variables are

invalid. For example, users typing individual commands at a terminal might want *date* to work if *LC\_MONETARY* is invalid as long as *LC\_TIME* is valid. Since these are conflicting reasonable alternatives, IEEE Std 1003.1-2001 leaves the results unspecified if the locale environment variables would not produce a complete locale matching the specification of the user.

The locale settings of individual categories cannot be truly independent and still guarantee correct results. For example, when collating two strings, characters must first be extracted from each string (governed by *LC\_CTYPE*) before being mapped to collating elements (governed by *LC\_COLLATE*) for comparison. That is, if *LC\_CTYPE* is causing parsing according to the rules of a large, multi-byte code set (potentially returning 20 000 or more distinct character codeset values), but *LC\_COLLATE* is set to handle only an 8-bit codeset with 256 distinct characters, meaningful results are obviously impossible.

The *LC\_MESSAGES* variable affects the language of messages generated by the standard utilities.

The description of the environment variable names starting with the characters “LC\_” acknowledges the fact that the interfaces presented may be extended as new international functionality is required. In the ISO C standard, names preceded by “LC\_” are reserved in the name space for future categories.

To avoid name clashes, new categories and environment variables are divided into two classifications: “implementation-independent” and “implementation-defined”.

Implementation-independent names will have the following format:

*LC\_NAME*

where *NAME* is the name of the new category and environment variable. Capital letters must be used for implementation-independent names.

Implementation-defined names must be in lowercase letters, as below:

*LC\_name*

### A.8.3 Other Environment Variables

#### COLUMNS, LINES

The default values for the number of column positions, *COLUMNS*, and screen height, *LINES*, are unspecified because historical implementations use different methods to determine values corresponding to the size of the screen in which the utility is run. This size is typically known to the implementation through the value of *TERM*, or by more elaborate methods such as extensions to the *stty* utility or knowledge of how the user is dynamically resizing windows on a bit-mapped display terminal. Users should not need to set these variables in the environment unless there is a specific reason to override the default behavior of the implementation, such as to display data in an area arbitrarily smaller than the terminal or window. Values for these variables that are not decimal integers greater than zero are implicitly undefined values; it is unnecessary to enumerate all of the possible values outside of the acceptable set.



**LOGNAME**

In most implementations, the value of such a variable is easily forged, so security-critical applications should rely on other means of determining user identity. *LOGNAME* is required to be constructed from the portable filename character set for reasons of interchange. No diagnostic condition is specified for violating this rule, and no requirement for enforcement exists. The intent of the requirement is that if extended characters are used, the “guarantee” of portability implied by a standard is void.

**PATH**

Many historical implementations of the Bourne shell do not interpret a trailing colon to represent the current working directory and are thus non-conforming. The C Shell and the KornShell conform to IEEE Std 1003.1-2001 on this point. The usual name of dot may also be used to refer to the current working directory.

Many implementations historically have used a default value of */bin* and */usr/bin* for the *PATH* variable. IEEE Std 1003.1-2001 does not mandate this default path be identical to that retrieved from *getconf \_CS\_PATH* because it is likely that the standardized utilities may be provided in another directory separate from the directories used by some historical applications.

**SHELL**

The *SHELL* variable names the preferred shell of the user; it is a guide to applications. There is no direct requirement that that shell conform to IEEE Std 1003.1-2001; that decision should rest with the user. It is the intention of the standard developers that alternative shells be permitted, if the user chooses to develop or acquire one. An operating system that builds its shell into the “kernel” in such a manner that alternative shells would be impossible does not conform to the spirit of IEEE Std 1003.1-2001.

**TZ**

The quoted form of the timezone variable allows timezone names of the form UTC+1 (or any name that contains the character plus (+)), the character minus (−), or digits), which may be appropriate for countries that do not have an official timezone name. It would be coded as <UTC+1>+1<UTC+2>, which would cause *std* to have a value of UTC+1 and *dst* a value of UTC+2, each with a length of 5 characters. This does not appear to conflict with any existing usage. The characters ‘<’ and ‘>’ were chosen for quoting because they are easier to parse visually than a quoting character that does not provide some sense of bracketing (and in a string like this, such bracketing is helpful). They were also chosen because they do not need special treatment when assigning to the *TZ* variable. Users are often confused by embedding quotes in a string. Because ‘<’ and ‘>’ are meaningful to the shell, the whole string would have to be quoted, but that is easily explained. (Parentheses would have presented the same problems.) Although the ‘>’ symbol could have been permitted in the string by either escaping it or doubling it, it seemed of little value to require that. This could be provided as an extension if there was a need. Timezone names of this new form lead to a requirement that the value of {\_POSIX\_TZNAME\_MAX} change from 3 to 6.

Since the *TZ* environment variable is usually inherited by all applications started by a user after the value of the *TZ* environment variable is changed and since many applications run using the C or POSIX locale, using characters that are not in the portable character set in the *std* and *dst* fields could cause unexpected results.

The format of the *TZ* environment variable is changed in Issue 6 to allow for the quoted form, as defined in previous versions of the ISO POSIX-1 standard.

2347 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/7 is applied, adding the *ctime\_r()* and  
 2348 *localtime\_r()* functions to the list of functions that use the TZ environment variable.

## 2349 **A.9 Regular Expressions**

2350 Rather than repeating the description of REs for each utility supporting REs, the standard  
 2351 developers preferred a common, comprehensive description of regular expressions in one place.  
 2352 The most common behavior is described here, and exceptions or extensions to this are  
 2353 documented for the respective utilities, as appropriate.

2354 The BRE corresponds to the *ed* or historical *grep* type, and the ERE corresponds to the historical  
 2355 *egrep* type (now *grep -E*).

2356 The text is based on the *ed* description and substantially modified, primarily to aid developers  
 2357 and others in the understanding of the capabilities and limitations of REs. Much of this was  
 2358 influenced by internationalization requirements.

2359 It should be noted that the definitions in this section do not cover the *tr* utility; the *tr* syntax does  
 2360 not employ REs.

2361 The specification of REs is particularly important to internationalization because pattern  
 2362 matching operations are very basic operations in business and other operations. The syntax and  
 2363 rules of REs are intended to be as intuitive as possible to make them easy to understand and use.  
 2364 The historical rules and behavior do not provide that capability to non-English language users,  
 2365 and do not provide the necessary support for commonly used characters and language  
 2366 constructs. It was necessary to provide extensions to the historical RE syntax and rules to  
 2367 accommodate other languages.

2368 As they are limited to bracket expressions, the rationale for these modifications is in the Base  
 2369 Definitions volume of IEEE Std 1003.1-2001, Section 9.3.5, RE Bracket Expression.

### 2370 **A.9.1 Regular Expression Definitions**

2371 It is possible to determine what strings correspond to subexpressions by recursively applying  
 2372 the leftmost longest rule to each subexpression, but only with the proviso that the overall match  
 2373 is leftmost longest. For example, matching "*\(ac\*\\)c\*d[ac]\*\1*" against *acdacaaa* matches  
 2374 *acdacaaa* (with *\1=a*); simply matching the longest match for "*\(ac\*\)*" would yield *\1=ac*, but  
 2375 the overall match would be smaller (*acdac*). Conceptually, the implementation must examine  
 2376 every possible match and among those that yield the leftmost longest total matches, pick the one  
 2377 that does the longest match for the leftmost subexpression, and so on. Note that this means that  
 2378 matching by subexpressions is context-dependent: a subexpression within a larger RE may  
 2379 match a different string from the one it would match as an independent RE, and two instances of  
 2380 the same subexpression within the same larger RE may match different lengths even in similar  
 2381 sequences of characters. For example, in the ERE "*(a.\*b)(a.\*b)*", the two identical  
 2382 subexpressions would match four and six characters, respectively, of *accbaccccb*.

2383 The definition of single character has been expanded to include also collating elements  
 2384 consisting of two or more characters; this expansion is applicable only when a bracket  
 2385 expression is included in the BRE or ERE. An example of such a collating element may be the  
 2386 Dutch *ij*, which collates as a 'y'. In some encodings, a ligature "i with j" exists as a character  
 2387 and would represent a single-character collating element. In another encoding, no such ligature  
 2388 exists, and the two-character sequence *ij* is defined as a multi-character collating element.  
 2389 Outside brackets, the *ij* is treated as a two-character RE and matches the same characters in a  
 2390 string. Historically, a bracket expression only matched a single character. The ISO POSIX-2: 1993  
 2391 standard required bracket expressions like "*[^[:lower:]]*" to match multi-character collating

elements such as "ij". However, this requirement led to behavior that many users did not expect and that could not feasibly be mimicked in user code, and it was rarely if ever implemented correctly. The current standard leaves it unspecified whether a bracket expression matches a multi-character collating element, allowing both historical and ISO POSIX-2:1993 standard implementations to conform.

Also, in the current standard, it is unspecified whether character class expressions like "[[:lower:]]" can include multi-character collating elements like "ij"; hence "[[:lower:]]" can match "ij", and "[^[:lower:]]" can fail to match "ij". Common practice is for a character class expression to match a collating element if it matches the collating element's first character.

## A.9.2 Regular Expression General Requirements

The definition of which sequence is matched when several are possible is based on the leftmost-longest rule historically used by deterministic recognizers. This rule is easier to define and describe, and arguably more useful, than the first-match rule historically used by non-deterministic recognizers. It is thought that dependencies on the choice of rule are rare; carefully contrived examples are needed to demonstrate the difference.

A formal expression of the leftmost-longest rule is:

The search is performed as if all possible suffixes of the string were tested for a prefix matching the pattern; the longest suffix containing a matching prefix is chosen, and the longest possible matching prefix of the chosen suffix is identified as the matching sequence.

Historically, most RE implementations only match lines, not strings. However, that is more an effect of the usage than of an inherent feature of REs themselves. Consequently, IEEE Std 1003.1-2001 does not regard <newline>s as special; they are ordinary characters, and both a period and a non-matching list can match them. Those utilities (like *grep*) that do not allow <newline>s to match are responsible for eliminating any <newline> from strings before matching against the RE. The *regcomp()* function, however, can provide support for such processing without violating the rules of this section.

Some implementations of *egrep* have had very limited flexibility in handling complex EREs. IEEE Std 1003.1-2001 does not attempt to define the complexity of a BRE or ERE, but does place a lower limit on it—any RE must be handled, as long as it can be expressed in 256 bytes or less. (Of course, this does not place an upper limit on the implementation.) There are historical programs using a non-deterministic-recognizer implementation that should have no difficulty with this limit. It is possible that a good approach would be to attempt to use the faster, but more limited, deterministic recognizer for simple expressions and to fall back on the non-deterministic recognizer for those expressions requiring it. Non-deterministic implementations must be careful to observe the rules on which match is chosen; the longest match, not the first match, starting at a given character is used.

The term “invalid” highlights a difference between this section and some others: IEEE Std 1003.1-2001 frequently avoids mandating of errors for syntax violations because they can be used by implementors to trigger extensions. However, the authors of the internationalization features of REs wanted to mandate errors for certain conditions to identify usage problems or non-portable constructs. These are identified within this rationale as appropriate. The remaining syntax violations have been left implicitly or explicitly undefined. For example, the BRE construct "\{1,2,3\}" does not comply with the grammar. A conforming application cannot rely on it producing an error nor matching the literal characters "\{1,2,3\}".

2438 The term “undefined” was used in favor of “unspecified” because many of the situations are  
 2439 considered errors on some implementations, and the standard developers considered that  
 2440 consistency throughout the section was preferable to mixing undefined and unspecified.

### 2441 **A.9.3 Basic Regular Expressions**

2442 There is no additional rationale provided for this section.

#### 2443 *A.9.3.1 BREs Matching a Single Character or Collating Element*

2444 There is no additional rationale provided for this section.

#### 2445 *A.9.3.2 BRE Ordinary Characters*

2446 There is no additional rationale provided for this section.

#### 2447 *A.9.3.3 BRE Special Characters*

2448 There is no additional rationale provided for this section.

#### 2449 *A.9.3.4 Periods in BREs*

2450 There is no additional rationale provided for this section.

#### 2451 *A.9.3.5 RE Bracket Expression*

2452 Range expressions are, historically, an integral part of REs. However, the requirements of  
 2453 “natural language behavior” and portability do conflict. In the POSIX locale, ranges must be  
 2454 treated according to the collating sequence and include such characters that fall within the range  
 2455 based on that collating sequence, regardless of character values. In other locales, ranges have  
 2456 unspecified behavior.

2457 Some historical implementations allow range expressions where the ending range point of one  
 2458 range is also the starting point of the next (for instance, " [a-m-o] "). This behavior should not  
 2459 be permitted, but to avoid breaking historical implementations, it is now *undefined* whether it is a  
 2460 valid expression and how it should be interpreted.

2461 Current practice in *awk* and *lex* is to accept escape sequences in bracket expressions as per the  
 2462 Base Definitions volume of IEEE Std 1003.1-2001, Table 5-1, Escape Sequences and Associated  
 2463 Actions, while the normal ERE behavior is to regard such a sequence as consisting of two  
 2464 characters. Allowing the *awk/lex* behavior in EREs would change the normal behavior in an  
 2465 unacceptable way; it is expected that *awk* and *lex* will decode escape sequences in EREs before  
 2466 passing them to *regcomp()* or comparable routines. Each utility describes the escape sequences it  
 2467 accepts as an exception to the rules in this section; the list is not the same, for historical reasons.

2468 As noted previously, the new syntax and rules have been added to accommodate other  
 2469 languages than English. The remainder of this section describes the rationale for these  
 2470 modifications.

2471 In the POSIX locale, a regular expression that starts with a range expression matches a set of  
 2472 strings that are contiguously sorted, but this is not necessarily true in other locales. For example,  
 2473 a French locale might have the following behavior:

```
2474 $ ls
2475 alpha Alpha estimé ESTIMÉ été eurêka
2476 $ ls [a-e] *
2477 alpha Alpha estimé eurêka
```

Such disagreements between matching and contiguous sorting are unavoidable because POSIX sorting cannot be implemented in terms of a deterministic finite-state automaton (DFA), but range expressions by design are implementable in terms of DFAs.

Historical implementations used native character order to interpret range expressions. The ISO POSIX-2:1993 standard instead required collating element order (CEO): the order that collating elements were specified between the **order\_start** and **order\_end** keywords in the *LC\_COLLATE* category of the current locale. CEO had some advantages in portability over the native character order, but it also had some disadvantages:

- CEO could not feasibly be mimicked in user code, leading to inconsistencies between POSIX matchers and matchers in popular user programs like Emacs, *ksh*, and Perl.
- CEO caused range expressions to match accented and capitalized letters contrary to many users' expectations. For example, "[a-e]" typically matched both 'E' and 'á' but neither 'A' nor 'é'.
- CEO was not consistent across implementations. In practice, CEO was often less portable than native character order. For example, it was common for the CEOs of two implementation-supplied locales to disagree, even if both locales were named "da\_DK".

Because of these problems, some implementations of regular expressions continued to use native character order. Others used the collation sequence, which is more consistent with sorting than either CEO or native order, but which departs further from the traditional POSIX semantics because it generally requires "[a-e]" to match either 'A' or 'E' but not both. As a result of this kind of implementation variation, programmers who wanted to write portable regular expressions could not rely on the ISO POSIX-2:1993 standard guarantees in practice.

While revising the standard, lengthy consideration was given to proposals to attack this problem by adding an API for querying the CEO to allow user-mode matchers, but none of these proposals had implementation experience and none achieved consensus. Leaving the standard alone was also considered, but rejected due to the problems described above.

The current standard leaves unspecified the behavior of a range expression outside the POSIX locale. This makes it clearer that conforming applications should avoid range expressions outside the POSIX locale, and it allows implementations and compatible user-mode matchers to interpret range expressions using native order, CEO, collation sequence, or other, more advanced techniques. The concerns which led to this change were raised in IEEE PASC interpretation 1003.2 #43 and others, and related to ambiguities in the specification of how multi-character collating elements should be handled in range expressions. These ambiguities had led to multiple interpretations of the specification, in conflicting ways, which led to varying implementations. As noted above, efforts were made to resolve the differences, but no solution has been found that would be specific enough to allow for portable software while not invalidating existing implementations.

The standard developers recognize that collating elements are important, such elements being common in several European languages; for example, 'ch' or 'll' in traditional Spanish; 'aa' in several Scandinavian languages. Existing internationalized implementations have processed, and continue to process, these elements in range expressions. Efforts are expected to continue in the future to find a way to define the behavior of these elements precisely and portably.

The ISO POSIX-2:1993 standard required "[b-a]" to be an invalid expression in the POSIX locale, but this requirement has been relaxed in this version of the standard so that "[b-a]" can instead be treated as a valid expression that does not match any string.

## 2523 A.9.3.6 BREs Matching Multiple Characters

2524 The limit of nine back-references to subexpressions in the RE is based on the use of a single-digit  
 2525 identifier; increasing this to multiple digits would break historical applications. This does not  
 2526 imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten  
 2527 subexpressions:

2528 `\(\(\(ab\) *c\) *d\) \ (ef\) * \ (gh\) \{2\} \ (ij\) * \ (kl\) * \ (mn\) * \ (op\) * \ (qr\) *`

2529 The standard developers regarded the common historical behavior, which supported "`\n*`", but  
 2530 not "`\n\{min,max\}`", "`\(...\)*`", or "`\(...\)\{min,max\}`", as a non-intentional  
 2531 result of a specific implementation, and they supported both duplication and interval  
 2532 expressions following subexpressions and back-references.

2533 The changes to the processing of the back-reference expression remove an unspecified or  
 2534 ambiguous behavior in the Shell and Utilities volume of IEEE Std 1003.1-2001, aligning it with  
 2535 the requirements specified for the *regcomp()* expression, and is the result of PASC Interpretation  
 2536 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.

## 2537 A.9.3.7 BRE Precedence

2538 There is no additional rationale provided for this section.

## 2539 A.9.3.8 BRE Expression Anchoring

2540 Often, the dollar sign is viewed as matching the ending <newline> in text files. This is not  
 2541 strictly true; the <newline> is typically eliminated from the strings to be matched, and the dollar  
 2542 sign matches the terminating null character.

2543 The ability of '`^`', '`$`', and '`*`' to be non-special in certain circumstances may be confusing to  
 2544 some programmers, but this situation was changed only in a minor way from historical practice  
 2545 to avoid breaking many historical scripts. Some consideration was given to making the use of  
 2546 the anchoring characters undefined if not escaped and not at the beginning or end of strings.  
 2547 This would cause a number of historical BREs, such as "`2^10`", "`$HOME`", and "`$1.35`", that  
 2548 relied on the characters being treated literally, to become invalid.

2549 However, one relatively uncommon case was changed to allow an extension used on some  
 2550 implementations. Historically, the BREs "`^foo`" and "`\(^foo\)`" did not match the same  
 2551 string, despite the general rule that subexpressions and entire BREs match the same strings. To  
 2552 increase consensus, IEEE Std 1003.1-2001 has allowed an extension on some implementations to  
 2553 treat these two cases in the same way by declaring that anchoring *may* occur at the beginning or  
 2554 end of a subexpression. Therefore, portable BREs that require a literal circumflex at the  
 2555 beginning or a dollar sign at the end of a subexpression must escape them. Note that a BRE such  
 2556 as "`a\(^bc\)`" will either match "`a^bc`" or nothing on different systems under the rules.

2557 ERE anchoring has been different from BRE anchoring in all historical systems. An unescaped  
 2558 anchor character has never matched its literal counterpart outside a bracket expression. Some  
 2559 implementations treated "`foo$bar`" as a valid expression that never matched anything; others  
 2560 treated it as invalid. IEEE Std 1003.1-2001 mandates the former, valid unmatched behavior.

2561 Some implementations have extended the BRE syntax to add alternation. For example, the  
 2562 subexpression "`\(foo$|bar\)`" would match either "`foo`" at the end of the string or "`bar`"  
 2563 anywhere. The extension is triggered by the use of the undefined "`|`" sequence. Because the  
 2564 BRE is undefined for portable scripts, the extending system is free to make other assumptions,  
 2565 such that the '`$`' represents the end-of-line anchor in the middle of a subexpression. If it were  
 2566 not for the extension, the '`$`' would match a literal dollar sign under the rules.

#### 2567 **A.9.4 Extended Regular Expressions**

2568 As with BREs, the standard developers decided to make the interpretation of escaped ordinary  
2569 characters undefined.

2570 The right parenthesis is not listed as an ERE special character because it is only special in the  
2571 context of a preceding left parenthesis. If found without a preceding left parenthesis, the right  
2572 parenthesis has no special meaning.

2573 The interval expression, " $\{m, n\}$ ", has been added to EREs. Historically, the interval expression  
2574 has only been supported in some ERE implementations. The standard developers estimated that  
2575 the addition of interval expressions to EREs would not decrease consensus and would also make  
2576 BREs more of a subset of EREs than in many historical implementations.

2577 It was suggested that, in addition to interval expressions, back-references ( $\backslash n$ ) should also be  
2578 added to EREs. This was rejected by the standard developers as likely to decrease consensus.

2579 In historical implementations, multiple duplication symbols are usually interpreted from left to  
2580 right and treated as additive. As an example, " $a^+b$ " matches zero or more instances of ' $a$ '  
2581 followed by a ' $b$ '. In IEEE Std 1003.1-2001, multiple duplication symbols are undefined; that is,  
2582 they cannot be relied upon for conforming applications. One reason for this is to provide some  
2583 scope for future enhancements.

2584 The precedence of operations differs between EREs and those in *lex*; in *lex*, for historical reasons,  
2585 interval expressions have a lower precedence than concatenation.

##### 2586 *A.9.4.1 EREs Matching a Single Character or Collating Element*

2587 There is no additional rationale provided for this section.

##### 2588 *A.9.4.2 ERE Ordinary Characters*

2589 There is no additional rationale provided for this section.

##### 2590 *A.9.4.3 ERE Special Characters*

2591 There is no additional rationale provided for this section.

##### 2592 *A.9.4.4 Periods in EREs*

2593 There is no additional rationale provided for this section.

##### 2594 *A.9.4.5 ERE Bracket Expression*

2595 There is no additional rationale provided for this section.

##### 2596 *A.9.4.6 EREs Matching Multiple Characters*

2597 There is no additional rationale provided for this section.

##### 2598 *A.9.4.7 ERE Alternation*

2599 There is no additional rationale provided for this section.

2600 A.9.4.8 *ERE Precedence*

2601 There is no additional rationale provided for this section.

2602 A.9.4.9 *ERE Expression Anchoring*

2603 There is no additional rationale provided for this section.

2604 **A.9.5 Regular Expression Grammar**

2605 The grammars are intended to represent the range of acceptable syntaxes available to  
 2606 conforming applications. There are instances in the text where undefined constructs are  
 2607 described; as explained previously, these allow implementation extensions. There is no intended  
 2608 requirement that an implementation extension must somehow fit into the grammars shown  
 2609 here.

2610 The BRE grammar does not permit `L_ANCHOR` or `R_ANCHOR` inside `"\"` and `"\"` (which  
 2611 implies that `'^'` and `'$'` are ordinary characters). This reflects the semantic limits on the  
 2612 application, as noted in the Base Definitions volume of IEEE Std 1003.1-2001, Section 9.3.8, BRE  
 2613 Expression Anchoring. Implementations are permitted to extend the language to interpret `'^'`  
 2614 and `'$'` as anchors in these locations, and as such, conforming applications cannot use  
 2615 unescaped `'^'` and `'$'` in positions inside `"\"` and `"\"` that might be interpreted as anchors.

2616 The ERE grammar does not permit several constructs that the Base Definitions volume of  
 2617 IEEE Std 1003.1-2001, Section 9.4.2, ERE Ordinary Characters and the Base Definitions volume of  
 2618 IEEE Std 1003.1-2001, Section 9.4.3, ERE Special Characters specify as having undefined results:

- 2619 • `ORD_CHAR` preceded by `'\"`
- 2620 • *ERE\_dupl\_symbol*(s) appearing first in an ERE, or immediately following `'|'`, `'^'`, or `'('`
- 2621 • `'{'` not part of a valid *ERE\_dupl\_symbol*
- 2622 • `'|'` appearing first or last in an ERE, or immediately following `'|'` or `'('`, or immediately  
 2623 preceding `')'`

2624 Implementations are permitted to extend the language to allow these. Conforming applications  
 2625 cannot use such constructs.

2626 A.9.5.1 *BRE/ERE Grammar Lexical Conventions*

2627 There is no additional rationale provided for this section.

2628 A.9.5.2 *RE and Bracket Expression Grammar*

2629 The removal of the *Back\_open\_paren Back\_close\_paren* option from the *nondupl\_RE* specification is  
 2630 the result of PASC Interpretation 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.  
 2631 Although the grammar required support for null subexpressions, this section does not describe  
 2632 the meaning of, and historical practice did not support, this construct.

2633 A.9.5.3 *ERE Grammar*

2634 There is no additional rationale provided for this section.



## 2635 **A.10 Directory Structure and Devices**

### 2636 **A.10.1 Directory Structure and Files**

2637 A description of the historical **/usr/tmp** was omitted, removing any concept of differences in  
 2638 emphasis between the **/** and **/usr** directories. The descriptions of **/bin**, **/usr/bin**, **/lib**, and **/usr/lib**  
 2639 were omitted because they are not useful for applications. In an early draft, a distinction was  
 2640 made between system and application directory usage, but this was not found to be useful.

2641 The directories **/** and **/dev** are included because the notion of a hierarchical directory structure is  
 2642 key to other information presented elsewhere in IEEE Std 1003.1-2001. In early drafts, it was  
 2643 argued that special devices and temporary files could conceivably be handled without a  
 2644 directory structure on some implementations. For example, the system could treat the characters  
 2645 **"/tmp"** as a special token that would store files using some non-POSIX file system structure.  
 2646 This notion was rejected by the standard developers, who required that all the files in this  
 2647 section be implemented via POSIX file systems.

2648 The **/tmp** directory is retained in IEEE Std 1003.1-2001 to accommodate historical applications  
 2649 that assume its availability. Implementations are encouraged to provide suitable directory  
 2650 names in the environment variable **TMPDIR** and applications are encouraged to use the contents  
 2651 of **TMPDIR** for creating temporary files.

2652 The standard files **/dev/null** and **/dev/tty** are required to be both readable and writable to allow  
 2653 applications to have the intended historical access to these files.

2654 The standard file **/dev/console** has been added for alignment with the Single UNIX Specification.

### 2655 **A.10.2 Output Devices and Terminal Types**

2656 There is no additional rationale provided for this section.

## 2657 **A.11 General Terminal Interface**

2658 If the implementation does not support this interface on any device types, it should behave as if  
 2659 it were being used on a device that is not a terminal device (in most cases *errno* will be set to  
 2660 [ENOTTY] on return from functions defined by this interface). This is based on the fact that  
 2661 many applications are written to run both interactively and in some non-interactive mode, and  
 2662 they adapt themselves at runtime. Requiring that they all be modified to test an environment  
 2663 variable to determine whether they should try to adapt is unnecessary. On a system that  
 2664 provides no general terminal interface, providing all the entry points as stubs that return  
 2665 [ENOTTY] (or an equivalent, as appropriate) has the same effect and requires no changes to the  
 2666 application.

2667 Although the needs of both interface implementors and application developers were addressed  
 2668 throughout IEEE Std 1003.1-2001, this section pays more attention to the needs of the latter. This  
 2669 is because, while many aspects of the programming interface can be hidden from the user by the  
 2670 application developer, the terminal interface is usually a large part of the user interface.  
 2671 Although to some extent the application developer can build missing features or work around  
 2672 inappropriate ones, the difficulties of doing that are greater in the terminal interface than  
 2673 elsewhere. For example, efficiency prohibits the average program from interpreting every  
 2674 character passing through it in order to simulate character erase, line kill, and so on. These  
 2675 functions should usually be done by the operating system, possibly at the interrupt level.

2676 The *tc\*()* functions were introduced as a way of avoiding the problems inherent in the  
 2677 traditional *ioctl()* function and in variants of it that were proposed. For example, *tcsetattr()* is

specified in place of the use of the TCSETA *ioctl()* command function. This allows specification of all the arguments in a manner consistent with the ISO C standard unlike the varying third argument of *ioctl()*, which is sometimes a pointer (to any of many different types) and sometimes an **int**.

The advantages of this new method include:

- It allows strict type checking.
- The direction of transfer of control data is explicit.
- Portable capabilities are clearly identified.
- The need for a general interface routine is avoided.
- Size of the argument is well-defined (there is only one type).

The disadvantages include:

- No historical implementation used the new method.
- There are many small routines instead of one general-purpose one.
- The historical parallel with *fcntl()* is broken.

The issue of modem control was excluded from IEEE Std 1003.1-2001 on the grounds that:

- It was concerned with setting and control of hardware timers.
- The appropriate timers and settings vary widely internationally.
- Feedback from European computer manufacturers indicated that this facility was not consistent with European needs and that specification of such a facility was not a requirement for portability.

## **A.11.1 Interface Characteristics**

### *A.11.1.1 Opening a Terminal Device File*

There is no additional rationale provided for this section.

### *A.11.1.2 Process Groups*

There is a potential race when the members of the foreground process group on a terminal leave that process group, either by exit or by changing process groups. After the last process exits the process group, but before the foreground process group ID of the terminal is changed (usually by a job control shell), it would be possible for a new process to be created with its process ID equal to the terminal's foreground process group ID. That process might then become the process group leader and accidentally be placed into the foreground on a terminal that was not necessarily its controlling terminal. As a result of this problem, the controlling terminal is defined to not have a foreground process group during this time.

The cases where a controlling terminal has no foreground process group occur when all processes in the foreground process group either terminate and are waited for or join other process groups via *setpgid()* or *setsid()*. If the process group leader terminates, this is the first case described; if it leaves the process group via *setpgid()*, this is the second case described (a process group leader cannot successfully call *setsid()*). When one of those cases causes a controlling terminal to have no foreground process group, it has two visible effects on applications. The first is the value returned by *tcgetpgrp()*. The second (which occurs only in the case where the process group leader terminates) is the sending of signals in response to special input characters. The intent of IEEE Std 1003.1-2001 is that no process group be wrongly

identified as the foreground process group by *tcgetpgrp()* or unintentionally receive signals because of placement into the foreground.

In 4.3 BSD, the old process group ID continues to be used to identify the foreground process group and is returned by the function equivalent to *tcgetpgrp()*. In that implementation it is possible for a newly created process to be assigned the same value as a process ID and then form a new process group with the same value as a process group ID. The result is that the new process group would receive signals from this terminal for no apparent reason, and IEEE Std 1003.1-2001 precludes this by forbidding a process group from entering the foreground in this way. It would be more direct to place part of the requirement made by the last sentence under *fork()*, but there is no convenient way for that section to refer to the value that *tcgetpgrp()* returns, since in this case there is no process group and thus no process group ID.

One possibility for a conforming implementation is to behave similarly to 4.3 BSD, but to prevent this reuse of the ID, probably in the implementation of *fork()*, as long as it is in use by the terminal.

Another possibility is to recognize when the last process stops using the terminal's foreground process group ID, which is when the process group lifetime ends, and to change the terminal's foreground process group ID to a reserved value that is never used as a process ID or process group ID. (See the definition of *process group lifetime* in the definitions section.) The process ID can then be reserved until the terminal has another foreground process group.

The 4.3 BSD implementation permits the leader (and only member) of the foreground process group to leave the process group by calling the equivalent of *setpgid()* and to later return, expecting to return to the foreground. There are no known application needs for this behavior, and IEEE Std 1003.1-2001 neither requires nor forbids it (except that it is forbidden for session leaders) by leaving it unspecified.

#### 2743 A.11.1.3 The Controlling Terminal

IEEE Std 1003.1-2001 does not specify a mechanism by which to allocate a controlling terminal. This is normally done by a system utility (such as *getty*) and is considered an administrative feature outside the scope of IEEE Std 1003.1-2001.

Historical implementations allocate controlling terminals on certain *open()* calls. Since *open()* is part of POSIX.1, its behavior had to be dealt with. The traditional behavior is not required because it is not very straightforward or flexible for either implementations or applications. However, because of its prevalence, it was not practical to disallow this behavior either. Thus, a mechanism was standardized to ensure portable, predictable behavior in *open()*.

Some historical implementations deallocate a controlling terminal on the last system-wide close. This behavior is neither required nor prohibited. Even on implementations that do provide this behavior, applications generally cannot depend on it due to its system-wide nature.

#### 2755 A.11.1.4 Terminal Access Control

The access controls described in this section apply only to a process that is accessing its controlling terminal. A process accessing a terminal that is not its controlling terminal is effectively treated the same as a member of the foreground process group. While this may seem unintuitive, note that these controls are for the purpose of job control, not security, and job control relates only to a process' controlling terminal. Normal file access permissions handle security.

If the process calling *read()* or *write()* is in a background process group that is orphaned, it is not desirable to stop the process group, as it is no longer under the control of a job control shell that could put it into the foreground again. Accordingly, calls to *read()* or *write()* functions by such

processes receive an immediate error return. This is different from 4.2 BSD, which kills orphaned processes that receive terminal stop signals.

The foreground/background/orphaned process group check performed by the terminal driver must be repeatedly performed until the calling process moves into the foreground or until the process group of the calling process becomes orphaned. That is, when the terminal driver determines that the calling process is in the background and should receive a job control signal, it sends the appropriate signal (SIGTTIN or SIGTTOU) to every process in the process group of the calling process and then it allows the calling process to immediately receive the signal. The latter is typically performed by blocking the process so that the signal is immediately noticed. Note, however, that after the process finishes receiving the signal and control is returned to the driver, the terminal driver must re-execute the foreground/background/orphaned process group check. The process may still be in the background, either because it was continued in the background by a job control shell, or because it caught the signal and did nothing.

The terminal driver repeatedly performs the foreground/background/orphaned process group checks whenever a process is about to access the terminal. In the case of *write()* or the control *tc\*()* functions, the check is performed at the entry of the function. In the case of *read()*, the check is performed not only at the entry of the function, but also after blocking the process to wait for input characters (if necessary). That is, once the driver has determined that the process calling the *read()* function is in the foreground, it attempts to retrieve characters from the input queue. If the queue is empty, it blocks the process waiting for characters. When characters are available and control is returned to the driver, the terminal driver must return to the repeated foreground/background/orphaned process group check again. The process may have moved from the foreground to the background while it was blocked waiting for input characters.

#### 2788 A.11.1.5 Input Processing and Reading Data

2789 There is no additional rationale provided for this section.

#### 2790 A.11.1.6 Canonical Mode Input Processing

2791 The term “character” is intended here. ERASE should erase the last character, not the last byte.  
2792 In the case of multi-byte characters, these two may be different.

2793 4.3 BSD has a WERASE character that erases the last “word” typed (but not any preceding  
2794 <blank>s or <tab>s). A word is defined as a sequence of non-<blank>s, with <tab>s counted as  
2795 <blank>s. Like ERASE, WERASE does not erase beyond the beginning of the line. This  
2796 WERASE feature has not been specified in POSIX.1 because it is difficult to define in the  
2797 international environment. It is only useful for languages where words are delimited by  
2798 <blank>s. In some ideographic languages, such as Japanese and Chinese, words are not  
2799 delimited at all. The WERASE character should presumably go back to the beginning of a  
2800 sentence in those cases; practically, this means it would not be used much for those languages.

2801 It should be noted that there is a possible inherent deadlock if the application and  
2802 implementation conflict on the value of {MAX\_CANON}. With ICANON set (if IXOFF is  
2803 enabled) and more than {MAX\_CANON} characters transmitted without a <linefeed>,  
2804 transmission will be stopped, the <linefeed> (or <carriage-return> when ICRLF is set) will never  
2805 arrive, and the *read()* will never be satisfied.

2806 An application should not set IXOFF if it is using canonical mode unless it knows that (even in  
2807 the face of a transmission error) the conditions described previously cannot be met or unless it is  
2808 prepared to deal with the possible deadlock in some other way, such as timeouts.

2809 It should also be noted that this can be made to happen in non-canonical mode if the trigger  
2810 value for sending IXOFF is less than VMIN and VTIME is zero.

## 2811 A.11.1.7 Non-Canonical Mode Input Processing

2812 Some points to note about MIN and TIME:

2813 1. The interactions of MIN and TIME are not symmetric. For example, when MIN>0 and  
 2814 TIME=0, TIME has no effect. However, in the opposite case where MIN=0 and TIME>0,  
 2815 both MIN and TIME play a role in that MIN is satisfied with the receipt of a single  
 2816 character.

2817 2. Also note that in case A (MIN>0, TIME>0), TIME represents an inter-character timer, while  
 2818 in case C (MIN=0, TIME>0), TIME represents a read timer.

2819 These two points highlight the dual purpose of the MIN /TIME feature. Cases A and B, where  
 2820 MIN>0, exist to handle burst-mode activity (for example, file transfer programs) where a  
 2821 program would like to process at least MIN characters at a time. In case A, the inter-character  
 2822 timer is activated by a user as a safety measure; in case B, it is turned off.

2823 Cases C and D exist to handle single-character timed transfers. These cases are readily adaptable  
 2824 to screen-based applications that need to know if a character is present in the input queue before  
 2825 refreshing the screen. In case C, the read is timed; in case D, it is not.

2826 Another important note is that MIN is always just a minimum. It does not denote a record  
 2827 length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20  
 2828 characters are returned to the user. In the special case of MIN=0, this still applies: if more than  
 2829 one character is available, they all will be returned immediately.

## 2830 A.11.1.8 Writing Data and Output Processing

2831 There is no additional rationale provided for this section.

## 2832 A.11.1.9 Special Characters

2833 There is no additional rationale provided for this section.

## 2834 A.11.1.10 Modem Disconnect

2835 There is no additional rationale provided for this section.

## 2836 A.11.1.11 Closing a Terminal Device File

2837 IEEE Std 1003.1-2001 does not specify that a *close()* on a terminal device file include the  
 2838 equivalent of a call to *tcflow(fd,TCOON)*.

2839 An implementation that discards output at the time *close()* is called after reporting the return  
 2840 value to the *write()* call that data was written does not conform with IEEE Std 1003.1-2001. An  
 2841 application has functions such as *tcdrain()*, *tcflush()*, and *tcflow()* available to obtain the detailed  
 2842 behavior it requires with respect to flushing of output.

2843 At the time of the last close on a terminal device, an application relinquishes any ability to exert  
 2844 flow control via *tcflow()*.

## 2845 **A.11.2 Parameters that Can be Set**

### 2846 *A.11.2.1 The termios Structure*

2847 This structure is part of an interface that, in general, retains the historic grouping of flags.  
 2848 Although a more optimal structure for implementations may be possible, the degree of change  
 2849 to applications would be significantly larger.

### 2850 *A.11.2.2 Input Modes*

2851 Some historical implementations treated a long break as multiple events, as many as one per  
 2852 character time. The wording in POSIX.1 explicitly prohibits this.

2853 Although the ISTRIP flag is normally superfluous with today's terminal hardware and software,  
 2854 it is historically supported. Therefore, applications may be using ISTRIP, and there is no  
 2855 technical problem with supporting this flag. Also, applications may wish to receive only 7-bit  
 2856 input bytes and may not be connected directly to the hardware terminal device (for example,  
 2857 when a connection traverses a network).

2858 Also, there is no requirement in general that the terminal device ensures that high-order bits  
 2859 beyond the specified character size are cleared. ISTRIP provides this function for 7-bit  
 2860 characters, which are common.

2861 In dealing with multi-byte characters, the consequences of a parity error in such a character, or in  
 2862 an escape sequence affecting the current character set, are beyond the scope of POSIX.1 and are  
 2863 best dealt with by the application processing the multi-byte characters.

### 2864 *A.11.2.3 Output Modes*

2865 POSIX.1 does not describe postprocessing of output to a terminal or detailed control of that from  
 2866 a conforming application. (That is, translation of <newline> to <carriage-return> followed by  
 2867 <linefeed> or <tab> processing.) There is nothing that a conforming application should do to its  
 2868 output for a terminal because that would require knowledge of the operation of the terminal. It  
 2869 is the responsibility of the operating system to provide postprocessing appropriate to the output  
 2870 device, whether it is a terminal or some other type of device.

2871 Extensions to POSIX.1 to control the type of postprocessing already exist and are expected to  
 2872 continue into the future. The control of these features is primarily to adjust the interface between  
 2873 the system and the terminal device so the output appears on the display correctly. This should  
 2874 be set up before use by any application.

2875 In general, both the input and output modes should not be set absolutely, but rather modified  
 2876 from the inherited state.

### 2877 *A.11.2.4 Control Modes*

2878 This section could be misread that the symbol "CSIZE" is a title in the **termios** *c\_cflag* field.  
 2879 Although it does serve that function, it is also a required symbol, as a literal reading of POSIX.1  
 2880 (and the caveats about typography) would indicate.

### 2881 *A.11.2.5 Local Modes*

2882 Non-canonical mode is provided to allow fast bursts of input to be read efficiently while still  
 2883 allowing single-character input.

2884 The ECHONL function historically has been in many implementations. Since there seems to be  
 2885 no technical problem with supporting ECHONL, it is included in POSIX.1 to increase consensus.

2886 The alternate behavior possible when ECHOK or ECHOE are specified with ICANON is  
 2887 permitted as a compromise depending on what the actual terminal hardware can do. Erasing  
 2888 characters and lines is preferred, but is not always possible.

#### 2889 A.11.2.6 Special Control Characters

2890 Permitting VMIN and VTIME to overlap with VEOF and VEOL was a compromise for historical  
 2891 implementations. Only when backwards-compatibility of object code is a serious concern to an  
 2892 implementor should an implementation continue this practice. Correct applications that work  
 2893 with the overlap (at the source level) should also work if it is not present, but not the reverse.

## 2894 A.12 Utility Conventions

### 2895 A.12.1 Utility Argument Syntax

2896 The standard developers considered that recent trends toward diluting the SYNOPSIS sections  
 2897 of historical reference pages to the equivalent of:

2898 `command [options] [operands]`

2899 were a disservice to the reader. Therefore, considerable effort was placed into rigorous  
 2900 definitions of all the command line arguments and their interrelationships. The relationships  
 2901 depicted in the synopses are normative parts of IEEE Std 1003.1-2001; this information is  
 2902 sometimes repeated in textual form, but that is only for clarity within context.

2903 The use of “undefined” for conflicting argument usage and for repeated usage of the same  
 2904 option is meant to prevent conforming applications from using conflicting arguments or  
 2905 repeated options unless specifically allowed (as is the case with *ls*, which allows simultaneous,  
 2906 repeated use of the *-C*, *-l*, and *-1* options). Many historical implementations will tolerate this  
 2907 usage, choosing either the first or the last applicable argument. This tolerance can continue, but  
 2908 conforming applications cannot rely upon it. (Other implementations may choose to print usage  
 2909 messages instead.)

2910 The use of “undefined” for conflicting argument usage also allows an implementation to make  
 2911 reasonable extensions to utilities where the implementor considers mutually-exclusive options  
 2912 according to IEEE Std 1003.1-2001 to have a sensible meaning and result.

2913 IEEE Std 1003.1-2001 does not define the result of a command when an option-argument or  
 2914 operand is not followed by ellipses and the application specifies more than one of that option-  
 2915 argument or operand. This allows an implementation to define valid (although non-standard)  
 2916 behavior for the utility when more than one such option or operand is specified.

2917 The following table summarizes the requirements for option-arguments:

|                                      | SYNOPSIS Shows:                |                                |                            |
|--------------------------------------|--------------------------------|--------------------------------|----------------------------|
|                                      | <i>-a arg</i>                  | <i>-barg</i>                   | <i>-c [arg]</i>            |
| Conforming application uses:         | <i>-a arg</i>                  | <i>-barg</i>                   | <i>-carg</i> or <i>-c</i>  |
| System supports:                     | <i>-a arg</i> and <i>-aarg</i> | <i>-b arg</i> and <i>-barg</i> | <i>-carg</i> and <i>-c</i> |
| Non-conforming applications may use: | <i>-aarg</i>                   | <i>-b arg</i>                  | N/A                        |

2925 Allowing <blank>s after an option (that is, placing an option and its option-argument into  
 2926 separate argument strings) when IEEE Std 1003.1-2001 does not require it encourages portability  
 2927 of users, while still preserving backwards-compatibility of scripts. Inserting <blank>s between

the option and the option-argument is preferred; however, historical usage has not been consistent in this area; therefore, <blank>s are required to be handled by all implementations, but implementations are also allowed to handle the historical syntax. Another justification for selecting the multiple-argument method was that the single-argument case is inherently ambiguous when the option-argument can legitimately be a null string.

IEEE Std 1003.1-2001 explicitly states that digits are permitted as operands and option-arguments. The lower and upper bounds for the values of the numbers used for operands and option-arguments were derived from the ISO C standard values for {LONG\_MIN} and {LONG\_MAX}. The requirement on the standard utilities is that numbers in the specified range do not cause a syntax error, although the specification of a number need not be semantically correct for a particular operand or option-argument of a utility. For example, the specification of:

```
dd obs=3000000000
```

would yield undefined behavior for the application and could be a syntax error because the number 3 000 000 000 is outside of the range −2 147 483 647 to +2 147 483 647. On the other hand:

```
dd obs=2000000000
```

may cause some error, such as “blocksize too large”, rather than a syntax error.

## A.12.2 Utility Syntax Guidelines

This section is based on the rules listed in the SVID. It was included for two reasons:

1. The individual utility descriptions in the Shell and Utilities volume of IEEE Std 1003.1-2001, Chapter 4, Utilities needed a set of common (although not universal) actions on which they could anchor their descriptions of option and operand syntax. Most of the standard utilities actually do use these guidelines, and many of their historical implementations use the *getopt()* function for their parsing. Therefore, it was simpler to cite the rules and merely identify exceptions.
2. Writers of conforming applications need suggested guidelines if the POSIX community is to avoid the chaos of historical UNIX system command syntax.

It is recommended that all *future* utilities and applications use these guidelines to enhance “user portability”. The fact that some historical utilities could not be changed (to avoid breaking historical applications) should not deter this future goal.

The voluntary nature of the guidelines is highlighted by repeated uses of the word *should* throughout. This usage should not be misinterpreted to imply that utilities that claim conformance in their OPTIONS sections do not always conform.

Guidelines 1 and 2 are offered as guidance for locales using Latin alphabets. No recommendations are made by IEEE Std 1003.1-2001 concerning utility naming in other locales.

In the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.9.1, Simple Commands, it is further stated that a command used in the Shell Command Language cannot be named with a trailing colon.

Guideline 3 was changed to allow alphanumeric characters (letters and digits) from the character set to allow compatibility with historical usage. Historical practice allows the use of digits wherever practical, and there are no portability issues that would prohibit the use of digits. In fact, from an internationalization viewpoint, digits (being non-language-dependent) are preferable over letters (a −2 is intuitively self-explanatory to any user, while in the −f *filename* the letter ‘f’ is a mnemonic aid only to speakers of Latin-based languages where “filename” happens to translate to a word that begins with ‘f’). Since guideline 3 still retains the word “single”, multi-digit options are not allowed. Instances of historical utilities that used them have



been marked obsolescent, with the numbers being changed from option names to option-arguments.

It was difficult to achieve a satisfactory solution to the problem of name space in option characters. When the standard developers desired to extend the historical *cc* utility to accept ISO C standard programs, they found that all of the portable alphabet was already in use by various vendors. Thus, they had to devise a new name, *c89* (now superseded by *c99*), rather than something like *cc -X*. There were suggestions that implementors be restricted to providing extensions through various means (such as using a plus sign as the option delimiter or using option characters outside the alphanumeric set) that would reserve all of the remaining alphanumeric characters for future POSIX standards. These approaches were resisted because they lacked the historical style of UNIX systems. Furthermore, if a vendor-provided option should become commonly used in the industry, it would be a candidate for standardization. It would be desirable to standardize such a feature using historical practice for the syntax (the semantics can be standardized with any syntax). This would not be possible if the syntax was one reserved for the vendor. However, since the standardization process may lead to minor changes in the semantics, it may prove to be better for a vendor to use a syntax that will not be affected by standardization.

Guideline 8 includes the concept of comma-separated lists in a single argument. It is up to the utility to parse such a list itself because *getopt()* just returns the single string. This situation was retained so that certain historical utilities would not violate the guidelines. Applications preparing for international use should be aware of an occasional problem with comma-separated lists: in some locales, the comma is used as the radix character. Thus, if an application is preparing operands for a utility that expects a comma-separated list, it should avoid generating non-integer values through one of the means that is influenced by setting the *LC\_NUMERIC* variable (such as *awk*, *bc*, *printf*, or *printf()*).

Applications calling any utility with a first operand starting with *'--'* should usually specify *--*, as indicated by Guideline 10, to mark the end of the options. This is true even if the SYNOPSIS in the Shell and Utilities volume of IEEE Std 1003.1-2001 does not specify any options; implementations may provide options as extensions to the Shell and Utilities volume of IEEE Std 1003.1-2001. The standard utilities that do not support Guideline 10 indicate that fact in the OPTIONS section of the utility description.

Guideline 11 was modified to clarify that the order of different options should not matter relative to one another. However, the order of repeated options that also have option-arguments may be significant; therefore, such options are required to be interpreted in the order that they are specified. The *make* utility is an instance of a historical utility that uses repeated options in which the order is significant. Multiple files are specified by giving multiple instances of the *-f* option; for example:

```
make -f common_header -f specific_rules target
```

Guideline 13 does not imply that all of the standard utilities automatically accept the operand *'--'* to mean standard input or output, nor does it specify the actions of the utility upon encountering multiple *'--'* operands. It simply says that, by default, *'--'* operands are not used for other purposes in the file reading or writing (but not when using *stat()*, *unlink()*, *touch*, and so on) utilities. All information concerning actual treatment of the *'--'* operand is found in the individual utility sections.

An area of concern was that as implementations mature, implementation-defined utilities and implementation-defined utility options will result. The idea was expressed that there needed to be a standard way, say an environment variable or some such mechanism, to identify implementation-defined utilities separately from standard utilities that may have the same name. It was decided that there already exist several ways of dealing with this situation and that

3022 it is outside of the scope to attempt to standardize in the area of non-standard items. A method  
3023 that exists on some historical implementations is the use of the so-called **/local/bin** or  
3024 **/usr/local/bin** directory to separate local or additional copies or versions of utilities. Another  
3025 method that is also used is to isolate utilities into completely separate domains. Still another  
3026 method to ensure that the desired utility is being used is to request the utility by its full  
3027 pathname. There are many approaches to this situation; the examples given above serve to  
3028 illustrate that there is more than one.

## 3029 **A.13 Headers**

### 3030 **A.13.1 Format of Entries**

3031 Each header reference page has a common layout of sections describing the interface. This layout  
3032 is similar to the manual page or “man” page format shipped with most UNIX systems, and each  
3033 header has sections describing the SYNOPSIS and DESCRIPTION. These are the two sections  
3034 that relate to conformance.

3035 Additional sections are informative, and add considerable information for the application  
3036 developer. APPLICATION USAGE sections provide additional caveats, issues, and  
3037 recommendations to the developer. RATIONALE sections give additional information on the  
3038 decisions made in defining the interface.

3039 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
3040 the future, and often cautions the developer to architect the code to account for a change in this  
3041 area. Note that a future directions statement should not be taken as a commitment to adopt a  
3042 feature or interface in the future.

3043 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
3044 changed.

3045 Option labels and margin markings in the page can be useful in guiding the application  
3046 developer.

3047 / *Rationale (Informative)*

3048 **Part B:**

3049 **System Interfaces**

3050 *The Open Group*

3051 *The Institute of Electrical and Electronics Engineers, Inc.*



# Rationale for System Interfaces

3052

## 3053 **B.1 Introduction**

### 3054 **B.1.1 Scope**

3055 Refer to Section A.1.1 (on page 3).

### 3056 **B.1.2 Conformance**

3057 Refer to Section A.2 (on page 9).

### 3058 **B.1.3 Normative References**

3059 There is no additional rationale provided for this section.

### 3060 **B.1.4 Change History**

3061 The change history is provided as an informative section, to track changes from previous issues  
3062 of IEEE Std 1003.1-2001.

3063 The following sections describe changes made to the System Interfaces volume of  
3064 IEEE Std 1003.1-2001 since Issue 5 of the base document. The CHANGE HISTORY section for  
3065 each entry details the technical changes that have been made to that entry from Issue 5. Changes  
3066 between earlier issues of the base document and Issue 5 are not included.

3067 The change history between Issue 5 and Issue 6 also lists the changes since the  
3068 ISO POSIX-1: 1996 standard.

#### 3069 **Changes from Issue 5 to Issue 6 (IEEE Std 1003.1-2001)**

3070 The following list summarizes the major changes that were made in the System Interfaces  
3071 volume of IEEE Std 1003.1-2001 from Issue 5 to Issue 6:

- 3072 • This volume of IEEE Std 1003.1-2001 is extensively revised so that it can be both an IEEE  
3073 POSIX Standard and an Open Group Technical Standard.
- 3074 • The POSIX System Interfaces requirements incorporate support of FIPS 151-2.
- 3075 • The POSIX System Interfaces requirements are updated to align with some features of the  
3076 Single UNIX Specification.
- 3077 • A RATIONALE section is added to each reference page.
- 3078 • Networking interfaces from the XNS, Issue 5.2 specification are incorporated.
- 3079 • IEEE Std 1003.1d-1999 is incorporated.
- 3080 • IEEE Std 1003.1j-2000 is incorporated.
- 3081 • IEEE Std 1003.1q-2000 is incorporated.
- 3082 • IEEE P1003.1a draft standard is incorporated.

- Existing functionality is aligned with the ISO / IEC 9899: 1999 standard.
- New functionality from the ISO / IEC 9899: 1999 standard is incorporated.
- IEEE PASC Interpretations are applied.
- The Open Group corrigenda and resolutions are applied.

### New Features in Issue 6

The functions first introduced in Issue 6 (over the Issue 5 Base document) are listed in the table below:

| New Functions in Issue 6 |                              |                        |
|--------------------------|------------------------------|------------------------|
| <i>acosf()</i>           | <i>catanhl()</i>             | <i>cprojf()</i>        |
| <i>acoshf()</i>          | <i>catanl()</i>              | <i>cprojl()</i>        |
| <i>acoshl()</i>          | <i>cbrtf()</i>               | <i>creal()</i>         |
| <i>acosl()</i>           | <i>cbrtl()</i>               | <i>crealf()</i>        |
| <i>asinf()</i>           | <i>ccos()</i>                | <i>creall()</i>        |
| <i>asinhf()</i>          | <i>ccosf()</i>               | <i>csin()</i>          |
| <i>asinhl()</i>          | <i>ccosh()</i>               | <i>csinf()</i>         |
| <i>asinl()</i>           | <i>ccoshf()</i>              | <i>csinh()</i>         |
| <i>atan2f()</i>          | <i>ccoshl()</i>              | <i>csinhf()</i>        |
| <i>atan2l()</i>          | <i>ccosl()</i>               | <i>csinhl()</i>        |
| <i>atanf()</i>           | <i>ceilf()</i>               | <i>csinl()</i>         |
| <i>atanhf()</i>          | <i>ceil()</i>                | <i>csqrt()</i>         |
| <i>atanhl()</i>          | <i>cexp()</i>                | <i>csqrtf()</i>        |
| <i>atanl()</i>           | <i>cexpf()</i>               | <i>csqrtl()</i>        |
| <i>atoll()</i>           | <i>cexpl()</i>               | <i>ctan()</i>          |
| <i>cabs()</i>            | <i>cimag()</i>               | <i>ctanf()</i>         |
| <i>cabsf()</i>           | <i>cimagf()</i>              | <i>ctanh()</i>         |
| <i>cabsl()</i>           | <i>cimagl()</i>              | <i>ctanhf()</i>        |
| <i>cacos()</i>           | <i>clock_getcpuclockid()</i> | <i>ctanhl()</i>        |
| <i>cacosf()</i>          | <i>clock_nanosleep()</i>     | <i>ctanl()</i>         |
| <i>cacosh()</i>          | <i>clog()</i>                | <i>erfcf()</i>         |
| <i>cacoshf()</i>         | <i>clogf()</i>               | <i>erfcl()</i>         |
| <i>cacoshl()</i>         | <i>clogl()</i>               | <i>erff()</i>          |
| <i>cacosl()</i>          | <i>conj()</i>                | <i>erfl()</i>          |
| <i>carg()</i>            | <i>conjf()</i>               | <i>exp2()</i>          |
| <i>cargf()</i>           | <i>conjl()</i>               | <i>exp2f()</i>         |
| <i>cargl()</i>           | <i>copysign()</i>            | <i>exp2l()</i>         |
| <i>casin()</i>           | <i>copysignf()</i>           | <i>expf()</i>          |
| <i>casinf()</i>          | <i>copysignl()</i>           | <i>expl()</i>          |
| <i>casinh()</i>          | <i>cosf()</i>                | <i>expm1f()</i>        |
| <i>casinhf()</i>         | <i>coshf()</i>               | <i>expm1l()</i>        |
| <i>casinhl()</i>         | <i>coshl()</i>               | <i>fabsf()</i>         |
| <i>casinl()</i>          | <i>cosl()</i>                | <i>fabsl()</i>         |
| <i>catan()</i>           | <i>cpow()</i>                | <i>fdim()</i>          |
| <i>catanf()</i>          | <i>cpowf()</i>               | <i>fdimf()</i>         |
| <i>catanh()</i>          | <i>cpowl()</i>               | <i>fdiml()</i>         |
| <i>catanhf()</i>         | <i>cproj()</i>               | <i>feclearexcept()</i> |

3129  
3130  
3131  
3132  
3133  
3134  
3135  
3136  
3137  
3138  
3139  
3140  
3141  
3142  
3143  
3144  
3145  
3146  
3147  
3148  
3149  
3150  
3151  
3152  
3153  
3154  
3155  
3156  
3157  
3158  
3159  
3160  
3161  
3162  
3163  
3164  
3165  
3166  
3167  
3168  
3169  
3170  
3171  
3172  
3173  
3174

| New Functions in Issue 6 |                          |                                                     |
|--------------------------|--------------------------|-----------------------------------------------------|
| <i>fegetenv()</i>        | <i>ldexpl()</i>          | <i>posix_fallocate()</i>                            |
| <i>fegetexceptflag()</i> | <i>lgammaf()</i>         | <i>posix_madvise()</i>                              |
| <i>fegetround()</i>      | <i>lgammal()</i>         | <i>posix_mem_offset()</i>                           |
| <i>feholdexcept()</i>    | <i>llabs()</i>           | <i>posix_memalign()</i>                             |
| <i>feraiseexcept()</i>   | <i>lldiv()</i>           | <i>posix_openpt()</i>                               |
| <i>fesetenv()</i>        | <i>llrint()</i>          | <i>posix_spawn()</i>                                |
| <i>fesetexceptflag()</i> | <i>llrintf()</i>         | <i>posix_spawn_file_actions_addclose()</i>          |
| <i>fesetround()</i>      | <i>llrintl()</i>         | <i>posix_spawn_file_actions_adddup2()</i>           |
| <i>fetestexcept()</i>    | <i>llround()</i>         | <i>posix_spawn_file_actions_addopen()</i>           |
| <i>feupdateenv()</i>     | <i>llroundf()</i>        | <i>posix_spawn_file_actions_destroy()</i>           |
| <i>floorf()</i>          | <i>llroundl()</i>        | <i>posix_spawn_file_actions_init()</i>              |
| <i>floorl()</i>          | <i>log10f()</i>          | <i>posix_spawnattr_destroy()</i>                    |
| <i>fma()</i>             | <i>log10l()</i>          | <i>posix_spawnattr_getflags()</i>                   |
| <i>maf()</i>             | <i>log1pf()</i>          | <i>posix_spawnattr_getpgroup()</i>                  |
| <i>fmal()</i>            | <i>log1pl()</i>          | <i>posix_spawnattr_getschedparam()</i>              |
| <i>fmax()</i>            | <i>log2()</i>            | <i>posix_spawnattr_getschedpolicy()</i>             |
| <i>fmaxf()</i>           | <i>log2f()</i>           | <i>posix_spawnattr_getsigdefault()</i>              |
| <i>fmaxl()</i>           | <i>log2l()</i>           | <i>posix_spawnattr_getsigmask()</i>                 |
| <i>fmin()</i>            | <i>logbf()</i>           | <i>posix_spawnattr_init()</i>                       |
| <i>fminf()</i>           | <i>logbl()</i>           | <i>posix_spawnattr_setflags()</i>                   |
| <i>fminl()</i>           | <i>logf()</i>            | <i>posix_spawnattr_setpgroup()</i>                  |
| <i>fmodf()</i>           | <i>logl()</i>            | <i>posix_spawnattr_setschedparam()</i>              |
| <i>fmodl()</i>           | <i>lrint()</i>           | <i>posix_spawnattr_setschedpolicy()</i>             |
| <i>fpclassify()</i>      | <i>lrintf()</i>          | <i>posix_spawnattr_setsigdefault()</i>              |
| <i>frexpf()</i>          | <i>lrintl()</i>          | <i>posix_spawnattr_setsigmask()</i>                 |
| <i>frexpl()</i>          | <i>lround()</i>          | <i>posix_spawnnp()</i>                              |
| <i>hypotf()</i>          | <i>lroundf()</i>         | <i>posix_trace_attr_destroy()</i>                   |
| <i>hypotl()</i>          | <i>lroundl()</i>         | <i>posix_trace_attr_getclockres()</i>               |
| <i>ilogbf()</i>          | <i>modff()</i>           | <i>posix_trace_attr_getcreatetime()</i>             |
| <i>ilogbl()</i>          | <i>modfl()</i>           | <i>posix_trace_attr_getgenversion()</i>             |
| <i>imaxabs()</i>         | <i>mq_timedreceive()</i> | <i>posix_trace_attr_getinherited()</i>              |
| <i>imaxdiv()</i>         | <i>mq_timedsend()</i>    | <i>posix_trace_attr_getlogfullpolicy()</i>          |
| <i>isblank()</i>         | <i>nan()</i>             | <i>posix_trace_attr_getlogsize()</i>                |
| <i>isfinite()</i>        | <i>nanf()</i>            | <i>posix_trace_attr_getmaxdatasize()</i>            |
| <i>isgreater()</i>       | <i>nanl()</i>            | <i>posix_trace_attr_getmaxsystemeventsizesize()</i> |
| <i>isgreaterequal()</i>  | <i>nearbyint()</i>       | <i>posix_trace_attr_getmaxusereventsizesize()</i>   |
| <i>isinf()</i>           | <i>nearbyintf()</i>      | <i>posix_trace_attr_getname()</i>                   |
| <i>isless()</i>          | <i>nearbyintl()</i>      | <i>posix_trace_attr_getstreamfullpolicy()</i>       |
| <i>islessequal()</i>     | <i>nextafterf()</i>      | <i>posix_trace_attr_getstreamsize()</i>             |
| <i>islessgreater()</i>   | <i>nextafterl()</i>      | <i>posix_trace_attr_init()</i>                      |
| <i>isnormal()</i>        | <i>nexttoward()</i>      | <i>posix_trace_attr_setinherited()</i>              |
| <i>isunordered()</i>     | <i>nexttowardf()</i>     | <i>posix_trace_attr_setlogfullpolicy()</i>          |
| <i>iswblank()</i>        | <i>nexttowardl()</i>     | <i>posix_trace_attr_setlogsize()</i>                |
| <i>ldexpf()</i>          | <i>posix_fadvise()</i>   | <i>posix_trace_create()</i>                         |

3175  
3176  
3177  
3178  
3179  
3180  
3181  
3182  
3183  
3184  
3185  
3186  
3187  
3188  
3189  
3190  
3191  
3192  
3193  
3194  
3195  
3196  
3197  
3198  
3199  
3200  
3201  
3202  
3203  
3204  
3205  
3206  
3207  
3208  
3209  
3210  
3211  
3212  
3213  
3214  
3215

### New Functions in Issue 6

|                                               |                                         |                     |
|-----------------------------------------------|-----------------------------------------|---------------------|
| <i>posix_trace_attr_setmaxdatasize()</i>      | <i>pthread_barrier_destroy()</i>        | <i>signbit()</i>    |
| <i>posix_trace_attr_setname()</i>             | <i>pthread_barrier_init()</i>           | <i>sinf()</i>       |
| <i>posix_trace_attr_setstreamfullpolicy()</i> | <i>pthread_barrier_wait()</i>           | <i>sinhf()</i>      |
| <i>posix_trace_attr_setstreamsize()</i>       | <i>pthread_barrierattr_destroy()</i>    | <i>sinhl()</i>      |
| <i>posix_trace_clear()</i>                    | <i>pthread_barrierattr_getpshared()</i> | <i>sinl()</i>       |
| <i>posix_trace_close()</i>                    | <i>pthread_barrierattr_init()</i>       | <i>sockatmark()</i> |
| <i>posix_trace_create_withlog()</i>           | <i>pthread_barrierattr_setpshared()</i> | <i>sqrtf()</i>      |
| <i>posix_trace_event()</i>                    | <i>pthread_condattr_getclock()</i>      | <i>sqrtl()</i>      |
| <i>posix_trace_eventid_equal()</i>            | <i>pthread_condattr_setclock()</i>      | <i>strerror_r()</i> |
| <i>posix_trace_eventid_get_name()</i>         | <i>pthread_getcpuclockid()</i>          | <i>strtoimax()</i>  |
| <i>posix_trace_eventid_open()</i>             | <i>pthread_mutex_timedlock()</i>        | <i>strtoll()</i>    |
| <i>posix_trace_eventset_add()</i>             | <i>pthread_rwlock_timedrdlock()</i>     | <i>strtoull()</i>   |
| <i>posix_trace_eventset_del()</i>             | <i>pthread_rwlock_timedwrlock()</i>     | <i>strtoumax()</i>  |
| <i>posix_trace_eventset_empty()</i>           | <i>pthread_setschedprio()</i>           | <i>tanf()</i>       |
| <i>posix_trace_eventset_fill()</i>            | <i>pthread_spin_destroy()</i>           | <i>tanhf()</i>      |
| <i>posix_trace_eventset_ismember()</i>        | <i>pthread_spin_init()</i>              | <i>tanhl()</i>      |
| <i>posix_trace_eventtypelist_getnext_id()</i> | <i>pthread_spin_lock()</i>              | <i>tanl()</i>       |
| <i>posix_trace_eventtypelist_rewind()</i>     | <i>pthread_spin_trylock()</i>           | <i>tgamma()</i>     |
| <i>posix_trace_flush()</i>                    | <i>pthread_spin_unlock()</i>            | <i>tgammaf()</i>    |
| <i>posix_trace_get_attr()</i>                 | <i>remainderf()</i>                     | <i>tgammal()</i>    |
| <i>posix_trace_get_filter()</i>               | <i>remainderl()</i>                     | <i>trunc()</i>      |
| <i>posix_trace_get_status()</i>               | <i>remquo()</i>                         | <i>truncf()</i>     |
| <i>posix_trace_getnext_event()</i>            | <i>remquof()</i>                        | <i>truncl()</i>     |
| <i>posix_trace_open()</i>                     | <i>remquol()</i>                        | <i>unsetenv()</i>   |
| <i>posix_trace_rewind()</i>                   | <i>rintf()</i>                          | <i>vfprintf()</i>   |
| <i>posix_trace_set_filter()</i>               | <i>rintl()</i>                          | <i>vfscanf()</i>    |
| <i>posix_trace_shutdown()</i>                 | <i>round()</i>                          | <i>vfwscanf()</i>   |
| <i>posix_trace_start()</i>                    | <i>roundf()</i>                         | <i>vprintf()</i>    |
| <i>posix_trace_stop()</i>                     | <i>roundl()</i>                         | <i>vscanf()</i>     |
| <i>posix_trace_timedgetnext_event()</i>       | <i>scalbln()</i>                        | <i>vsprintf()</i>   |
| <i>posix_trace_trid_eventid_open()</i>        | <i>scalblnf()</i>                       | <i>vsscanf()</i>    |
| <i>posix_trace_trygetnext_event()</i>         | <i>scalblnl()</i>                       | <i>vswscanf()</i>   |
| <i>posix_typed_mem_get_info()</i>             | <i>scalbn()</i>                         | <i>vwscanf()</i>    |
| <i>posix_typed_mem_open()</i>                 | <i>scalbnf()</i>                        | <i>wcstoimax()</i>  |
| <i>powf()</i>                                 | <i>scalbnl()</i>                        | <i>wcstoll()</i>    |
| <i>powl()</i>                                 | <i>sem_timedwait()</i>                  | <i>wcstoull()</i>   |
| <i>pselect()</i>                              | <i>setgid()</i>                         | <i>wcstoumax()</i>  |
| <i>pthread_attr_getstack()</i>                | <i>setenv()</i>                         |                     |
| <i>pthread_attr_setstack()</i>                | <i>setuid()</i>                         |                     |



3216           The following new headers are introduced in Issue 6:

3217

3218

3219

3220

3221

| New Headers in Issue 6 |             |            |
|------------------------|-------------|------------|
| <complex.h>            | <spawn.h>   | <tgmath.h> |
| <fenv.h>               | <stdbool.h> | <trace.h>  |
| <net/if.h>             | <stdint.h>  |            |

The following table lists the functions and symbols from the XSI extension. These are new since the ISO POSIX-1: 1996 standard.

| New XSI Functions and Symbols in Issue 6 |                             |                                              |                             |
|------------------------------------------|-----------------------------|----------------------------------------------|-----------------------------|
| <code>_longjmp()</code>                  | <code>getcontext()</code>   | <code>msgget()</code>                        | <code>setstate()</code>     |
| <code>_setjmp()</code>                   | <code>getdate()</code>      | <code>msgrcv()</code>                        | <code>setutxent()</code>    |
| <code>_tolower()</code>                  | <code>getgrent()</code>     | <code>msgsnd()</code>                        | <code>shmat()</code>        |
| <code>_toupper()</code>                  | <code>gethostid()</code>    | <code>nftw()</code>                          | <code>shmctl()</code>       |
| <code>a64l()</code>                      | <code>getitimer()</code>    | <code>nice()</code>                          | <code>shmdt()</code>        |
| <code>basename()</code>                  | <code>getpgid()</code>      | <code>nl_langinfo()</code>                   | <code>shmget()</code>       |
| <code>bcmp()</code>                      | <code>getpmsg()</code>      | <code>nrnd48()</code>                        | <code>sigaltstack()</code>  |
| <code>bcopy()</code>                     | <code>getpriority()</code>  | <code>openlog()</code>                       | <code>sighold()</code>      |
| <code>bzero()</code>                     | <code>getpwent()</code>     | <code>poll()</code>                          | <code>sigignore()</code>    |
| <code>catclose()</code>                  | <code>getrlimit()</code>    | <code>posix_openpt()</code>                  | <code>siginterrupt()</code> |
| <code>catgets()</code>                   | <code>getrusage()</code>    | <code>pread()</code>                         | <code>sigpause()</code>     |
| <code>catopen()</code>                   | <code>getsid()</code>       | <code>pthread_attr_getguardsize()</code>     | <code>sigrelse()</code>     |
| <code>closelog()</code>                  | <code>getsubopt()</code>    | <code>pthread_attr_setguardsize()</code>     | <code>sigset()</code>       |
| <code>crypt()</code>                     | <code>gettimeofday()</code> | <code>pthread_attr_setstack()</code>         | <code>srand48()</code>      |
| <code>daylight</code>                    | <code>getutxent()</code>    | <code>pthread_getconcurrency()</code>        | <code>srandom()</code>      |
| <code>dbm_clearerr()</code>              | <code>getutxid()</code>     | <code>pthread_mutexattr_gettype()</code>     | <code>statvfs()</code>      |
| <code>dbm_close()</code>                 | <code>getutxline()</code>   | <code>pthread_mutexattr_settype()</code>     | <code>strcasecmp()</code>   |
| <code>dbm_delete()</code>                | <code>getwd()</code>        | <code>pthread_rwlockattr_init()</code>       | <code>strdup()</code>       |
| <code>dbm_error()</code>                 | <code>grantpt()</code>      | <code>pthread_rwlockattr_setpshared()</code> | <code>strfmon()</code>      |
| <code>dbm_fetch()</code>                 | <code>hcreate()</code>      | <code>pthread_setconcurrency()</code>        | <code>strncasecmp()</code>  |
| <code>dbm_firstkey()</code>              | <code>hdestroy()</code>     | <code>ptsname()</code>                       | <code>strptime()</code>     |
| <code>dbm_nextkey()</code>               | <code>hsearch()</code>      | <code>putenv()</code>                        | <code>swab()</code>         |
| <code>dbm_open()</code>                  | <code>iconv()</code>        | <code>pututxline()</code>                    | <code>swapcontext()</code>  |
| <code>dbm_store()</code>                 | <code>iconv_close()</code>  | <code>pwrite()</code>                        | <code>sync()</code>         |
| <code>dirname()</code>                   | <code>iconv_open()</code>   | <code>random()</code>                        | <code>syslog()</code>       |
| <code>dldclose()</code>                  | <code>index()</code>        | <code>readv()</code>                         | <code>tcgetsid()</code>     |
| <code>dlderror()</code>                  | <code>initstate()</code>    | <code>realpath()</code>                      | <code>tdelete()</code>      |
| <code>dlopen()</code>                    | <code>insque()</code>       | <code>remque()</code>                        | <code>telldir()</code>      |
| <code>dlsym()</code>                     | <code>isascii()</code>      | <code>rindex()</code>                        | <code>tempnam()</code>      |
| <code>drand48()</code>                   | <code>jrand48()</code>      | <code>seed48()</code>                        | <code>tfind()</code>        |
| <code>ecvt()</code>                      | <code>killpg()</code>       | <code>seekdir()</code>                       | <code>timezone</code>       |
| <code>encrypt()</code>                   | <code>l64a()</code>         | <code>semctl()</code>                        | <code>toascii()</code>      |
| <code>endgrent()</code>                  | <code>lchown()</code>       | <code>semget()</code>                        | <code>truncate()</code>     |
| <code>endpwent()</code>                  | <code>lcong48()</code>      | <code>semop()</code>                         | <code>tsearch()</code>      |
| <code>endutxent()</code>                 | <code>lfind()</code>        | <code>setcontext()</code>                    | <code>twalk()</code>        |
| <code>erand48()</code>                   | <code>lockf()</code>        | <code>setgrent()</code>                      | <code>ulimit()</code>       |
| <code>fchdir()</code>                    | <code>lrnd48()</code>       | <code>setitimer()</code>                     | <code>unlockpt()</code>     |
| <code>fcvt()</code>                      | <code>lsearch()</code>      | <code>setkey()</code>                        | <code>utimes()</code>       |
| <code>ffs()</code>                       | <code>makecontext()</code>  | <code>setlogmask()</code>                    | <code>waitid()</code>       |
| <code>fmtmsg()</code>                    | <code>memccpy()</code>      | <code>setpggrp()</code>                      | <code>wcs wcs()</code>      |
| <code>fstatvfs()</code>                  | <code>mknod()</code>        | <code>setpriority()</code>                   | <code>wcswidth()</code>     |
| <code>ftime()</code>                     | <code>mkstemp()</code>      | <code>setpwent()</code>                      | <code>wcwidth()</code>      |
| <code>ftok()</code>                      | <code>mktemp()</code>       | <code>setregid()</code>                      | <code>writev()</code>       |
| <code>ftw()</code>                       | <code>mrnd48()</code>       | <code>setreuid()</code>                      |                             |
| <code>gcvt()</code>                      | <code>msgctl()</code>       | <code>setrlimit()</code>                     |                             |

The following table lists the headers from the XSI extension. These are new since the ISO POSIX-1: 1996 standard.

| New XSI Headers in Issue 6 |                  |                 |
|----------------------------|------------------|-----------------|
| <cpio.h>                   | <poll.h>         | <sys/statvfs.h> |
| <dlfcn.h>                  | <search.h>       | <sys/time.h>    |
| <fmtmsg.h>                 | <strings.h>      | <sys/timeb.h>   |
| <ftw.h>                    | <stropts.h>      | <sys/uio.h>     |
| <iconv.h>                  | <sys/ipc.h>      | <syslog.h>      |
| <langinfo.h>               | <sys/mman.h>     | <ucontext.h>    |
| <libgen.h>                 | <sys/msg.h>      | <ulimit.h>      |
| <monetary.h>               | <sys/resource.h> | <utmpx.h>       |
| <ndbm.h>                   | <sys/sem.h>      |                 |
| <nl_types.h>               | <sys/shm.h>      |                 |

### **B.1.5 Terminology**

Refer to Section A.1.4 (on page 5).

### **B.1.6 Definitions**

Refer to Section A.3 (on page 13).

### **B.1.7 Relationship to Other Formal Standards**

There is no additional rationale provided for this section.

### **B.1.8 Portability**

Refer to Section A.1.5 (on page 8).

#### **B.1.8.1 Codes**

Refer to Section A.1.5.1 (on page 8).

### **B.1.9 Format of Entries**

Each system interface reference page has a common layout of sections describing the interface. This layout is similar to the manual page or “man” page format shipped with most UNIX systems, and each header has sections describing the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS. These are the four sections that relate to conformance.

Additional sections are informative, and add considerable information for the application developer. EXAMPLES sections provide example usage. APPLICATION USAGE sections provide additional caveats, issues, and recommendations to the developer. RATIONALE sections give additional information on the decisions made in defining the interface.

FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in the future, and often cautions the developer to architect the code to account for a change in this area. Note that a future directions statement should not be taken as a commitment to adopt a feature or interface in the future.

The CHANGE HISTORY section describes when the interface was introduced, and how it has changed.

Option labels and margin markings in the page can be useful in guiding the application developer.

## B.2 General Information

### B.2.1 Use and Implementation of Functions

The information concerning the use of functions was adapted from a description in the ISO C standard. Here is an example of how an application program can protect itself from functions that may or may not be macros, rather than true functions:

The `atoi()` function may be used in any of several ways:

- By use of its associated header (possibly generating a macro expansion):

```
#include <stdlib.h>
/* ... */
i = atoi(str);
```

- By use of its associated header (assuredly generating a true function call):

```
#include <stdlib.h>
#undef atoi
/* ... */
i = atoi(str);
```

or:

```
#include <stdlib.h>
/* ... */
i = (atoi) (str);
```

- By explicit declaration:

```
extern int atoi (const char *);
/* ... */
i = atoi(str);
```

- By implicit declaration:

```
/* ... */
i = atoi(str);
```

(Assuming no function prototype is in scope. This is not allowed by the ISO C standard for functions with variable arguments; furthermore, parameter type conversion “widening” is subject to different rules in this case.)

Note that the ISO C standard reserves names starting with ‘\_’ for the compiler. Therefore, the compiler could, for example, implement an intrinsic, built-in function `_asm_builtin_atoi()`, which it recognized and expanded into inline assembly code. Then, in `<stdlib.h>`, there could be the following:

```
#define atoi(X) _asm_builtin_atoi(X)
```

The user’s “normal” call to `atoi()` would then be expanded inline, but the implementor would also be required to provide a callable function named `atoi()` for use when the application requires it; for example, if its address is to be stored in a function pointer variable.

## 3349 **B.2.2 The Compilation Environment**

### 3350 *B.2.2.1 POSIX.1 Symbols*

3351 This and the following section address the issue of “name space pollution”. The ISO C standard  
 3352 requires that the name space beyond what it reserves not be altered except by explicit action of  
 3353 the application writer. This section defines the actions to add the POSIX.1 symbols for those  
 3354 headers where both the ISO C standard and POSIX.1 need to define symbols, and also where the  
 3355 XSI extension extends the base standard.

3356 When headers are used to provide symbols, there is a potential for introducing symbols that the  
 3357 application writer cannot predict. Ideally, each header should only contain one set of symbols,  
 3358 but this is not practical for historical reasons. Thus, the concept of feature test macros is  
 3359 included. Two feature test macros are explicitly defined by IEEE Std 1003.1-2001; it is expected  
 3360 that future revisions may add to this.

3361 **Note:** Feature test macros allow an application to announce to the implementation its desire to have  
 3362 certain symbols and prototypes exposed. They should not be confused with the version test  
 3363 macros and constants for options in `<unistd.h>` which are the implementation’s way of  
 3364 announcing functionality to the application.

3365 It is further intended that these feature test macros apply only to the headers specified by  
 3366 IEEE Std 1003.1-2001. Implementations are expressly permitted to make visible symbols not  
 3367 specified by IEEE Std 1003.1-2001, within both POSIX.1 and other headers, under the control of  
 3368 feature test macros that are not defined by IEEE Std 1003.1-2001.

### 3369 **The `_POSIX_C_SOURCE` Feature Test Macro**

3370 Since `_POSIX_SOURCE` specified by the POSIX.1-1990 standard did not have a value associated  
 3371 with it, the `_POSIX_C_SOURCE` macro replaces it, allowing an application to inform the system  
 3372 of the revision of the standard to which it conforms. This symbol will allow implementations to  
 3373 support various revisions of IEEE Std 1003.1-2001 simultaneously. For instance, when either  
 3374 `_POSIX_SOURCE` is defined or `_POSIX_C_SOURCE` is defined as 1, the system should make  
 3375 visible the same name space as permitted and required by the POSIX.1-1990 standard. When  
 3376 `_POSIX_C_SOURCE` is defined, the state of `_POSIX_SOURCE` is completely irrelevant.

3377 It is expected that C bindings to future POSIX standards will define new values for  
 3378 `_POSIX_C_SOURCE`, with each new value reserving the name space for that new standard, plus  
 3379 all earlier POSIX standards.

### 3380 **The `_XOPEN_SOURCE` Feature Test Macro**

3381 The feature test macro `_XOPEN_SOURCE` is provided as the announcement mechanism for the  
 3382 application that it requires functionality from the Single UNIX Specification. `_XOPEN_SOURCE`  
 3383 must be defined to the value 600 before the inclusion of any header to enable the functionality in  
 3384 the Single UNIX Specification. Its definition subsumes the use of `_POSIX_SOURCE` and  
 3385 `_POSIX_C_SOURCE`.

3386 An extract of code from a conforming application, that appears before any **#include** statements,  
 3387 is given below:

```
3388 #define _XOPEN_SOURCE 600 /* Single UNIX Specification, Version 3 */
3389 #include ...
```

3390 Note that the definition of `_XOPEN_SOURCE` with the value 600 makes the definition of  
 3391 `_POSIX_C_SOURCE` redundant and it can safely be omitted.

## 3392 B.2.2.2 The Name Space

3393 The reservation of identifiers is paraphrased from the ISO C standard. The text is included  
 3394 because it needs to be part of IEEE Std 1003.1-2001, regardless of possible changes in future  
 3395 versions of the ISO C standard.

3396 These identifiers may be used by implementations, particularly for feature test macros.  
 3397 Implementations should not use feature test macro names that might be reasonably used by a  
 3398 standard.

3399 Including headers more than once is a reasonably common practice, and it should be carried  
 3400 forward from the ISO C standard. More significantly, having definitions in more than one  
 3401 header is explicitly permitted. Where the potential declaration is “benign” (the same definition  
 3402 twice) the declaration can be repeated, if that is permitted by the compiler. (This is usually true  
 3403 of macros, for example.) In those situations where a repetition is not benign (for example,  
 3404 **typedefs**), conditional compilation must be used. The situation actually occurs both within the  
 3405 ISO C standard and within POSIX.1: **time\_t** should be in **<sys/types.h>**, and the ISO C standard  
 3406 mandates that it be in **<time.h>**.

3407 The area of name space pollution *versus* additions to structures is difficult because of the macro  
 3408 structure of C. The following discussion summarizes all the various problems with and  
 3409 objections to the issue.

3410 Note the phrase “user-defined macro”. Users are not permitted to define macro names (or any  
 3411 other name) beginning with “\_**[A-Z\_]**”. Thus, the conflict cannot occur for symbols reserved  
 3412 to the vendor’s name space, and the permission to add fields automatically applies, without  
 3413 qualification, to those symbols.

3414 1. Data structures (and unions) need to be defined in headers by implementations to meet  
 3415 certain requirements of POSIX.1 and the ISO C standard.

3416 2. The structures defined by POSIX.1 are typically minimal, and any practical  
 3417 implementation would wish to add fields to these structures either to hold additional  
 3418 related information or for backwards-compatibility (or both). Future standards (and *de*  
 3419 *facto* standards) would also wish to add to these structures. Issues of field alignment make  
 3420 it impractical (at least in the general case) to simply omit fields when they are not defined  
 3421 by the particular standard involved.

3422 The **dirent** structure is an example of such a minimal structure (although one could argue  
 3423 about whether the other fields need visible names). The **st\_rdev** field of most  
 3424 implementations’ **stat** structure is a common example where extension is needed and  
 3425 where a conflict could occur.

3426 3. Fields in structures are in an independent name space, so the addition of such fields  
 3427 presents no problem to the C language itself in that such names cannot interact with  
 3428 identically named user symbols because access is qualified by the specific structure name.

3429 4. There is an exception to this: macro processing is done at a lexical level. Thus, symbols  
 3430 added to a structure might be recognized as user-provided macro names at the location  
 3431 where the structure is declared. This only can occur if the user-provided name is declared  
 3432 as a macro before the header declaring the structure is included. The user’s use of the name  
 3433 after the declaration cannot interfere with the structure because the symbol is hidden and  
 3434 only accessible through access to the structure. Presumably, the user would not declare  
 3435 such a macro if there was an intention to use that field name.

3436 5. Macros from the same or a related header might use the additional fields in the structure,  
 3437 and those field names might also collide with user macros. Although this is a less frequent  
 3438 occurrence, since macros are expanded at the point of use, no constraint on the order of use

of names can apply.

6. An “obvious” solution of using names in the reserved name space and then redefining them as macros when they should be visible does not work because this has the effect of exporting the symbol into the general name space. For example, given a (hypothetical) system-provided header **<h.h>**, and two parts of a C program in **a.c** and **b.c**, in header **<h.h>**:

```
struct foo {
 int __i;
}

#ifdef _FEATURE_TEST
#define i __i;
#endif
```

In file **a.c**:

```
#include h.h
extern int i;
...
```

In file **b.c**:

```
extern int i;
...
```

The symbol that the user thinks of as *i* in both files has an external name of `__i` in **a.c**; the same symbol *i* in **b.c** has an external name *i* (ignoring any hidden manipulations the compiler might perform on the names). This would cause a mysterious name resolution problem when **a.o** and **b.o** are linked.

Simply avoiding definition then causes alignment problems in the structure.

A structure of the form:

```
struct foo {
 union {
 int __i;
#ifdef _FEATURE_TEST
 int i;
#endif
 } __ii;
}
```

does not work because the name of the logical field *i* is `__ii.i`, and introduction of a macro to restore the logical name immediately reintroduces the problem discussed previously (although its manifestation might be more immediate because a syntax error would result if a recursive macro did not cause it to fail first).

7. A more workable solution would be to declare the structure:

```
struct foo {
#ifdef _FEATURE_TEST
 int i;
#else
 int __i;
#endif
}
```

However, if a macro (particularly one required by a standard) is to be defined that uses this field, two must be defined: one that uses *i*, the other that uses *\_\_i*. If more than one additional field is used in a macro and they are conditional on distinct combinations of features, the complexity goes up as  $2^n$ .

All this leaves a difficult situation: vendors must provide very complex headers to deal with what is conceptually simple and safe—adding a field to a structure. It is the possibility of user-provided macros with the same name that makes this difficult.

Several alternatives were proposed that involved constraining the user's access to part of the name space available to the user (as specified by the ISO C standard). In some cases, this was only until all the headers had been included. There were two proposals discussed that failed to achieve consensus:

1. Limiting it for the whole program.
2. Restricting the use of identifiers containing only uppercase letters until after all system headers had been included. It was also pointed out that because macros might wish to access fields of a structure (and macro expansion occurs totally at point of use) restricting names in this way would not protect the macro expansion, and thus the solution was inadequate.

It was finally decided that reservation of symbols would occur, but as constrained.

The current wording also allows the addition of fields to a structure, but requires that user macros of the same name not interfere. This allows vendors to do one of the following:

- Not create the situation (do not extend the structures with user-accessible names or use the solution in (7) above)
- Extend their compilers to allow some way of adding names to structures and macros safely

There are at least two ways that the compiler might be extended: add new preprocessor directives that turn off and on macro expansion for certain symbols (without changing the value of the macro) and a function or lexical operation that suppresses expansion of a word. The latter seems more flexible, particularly because it addresses the problem in macros as well as in declarations.

The following seems to be a possible implementation extension to the C language that will do this: any token that during macro expansion is found to be preceded by three '#' symbols shall not be further expanded in exactly the same way as described for macros that expand to their own name as in Section 3.8.3.4 of the ISO C standard. A vendor may also wish to implement this as an operation that is lexically a function, which might be implemented as:

```
#define __safe_name(x) ###x
```

Using a function notation would insulate vendors from changes in standards until such a functionality is standardized (if ever). Standardization of such a function would be valuable because it would then permit third parties to take advantage of it portably in software they may



3521 supply.

3522 The symbols that are “explicitly permitted, but not required by IEEE Std 1003.1-2001” include  
 3523 those classified below. (That is, the symbols classified below might, but are not required to, be  
 3524 present when `_POSIX_C_SOURCE` is defined to have the value 200112L.)

- 3525 • Symbols in `<limits.h>` and `<unistd.h>` that are defined to indicate support for options or  
 3526 limits that are constant at compile-time
- 3527 • Symbols in the name space reserved for the implementation by the ISO C standard
- 3528 • Symbols in a name space reserved for a particular type of extension (for example, type names  
 3529 ending with `_t` in `<sys/types.h>`)
- 3530 • Additional members of structures or unions whose names do not reduce the name space  
 3531 reserved for applications

3532 Since both implementations and future revisions of IEEE Std 1003.1 and other POSIX standards  
 3533 may use symbols in the reserved spaces described in these tables, there is a potential for name  
 3534 space clashes. To avoid future name space clashes when adding symbols, implementations  
 3535 should not use the `posix_`, `POSIX_`, or `_POSIX_` prefixes.

3536 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/2 is applied, deleting the entries `POSIX_`,  
 3537 `_POSIX_`, and `posix_` from the column of allowed name space prefixes for use by an  
 3538 implementation in the first table. The presence of these prefixes was contradicting later text  
 3539 which states that: “The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by Shell and  
 3540 Utilities volume of IEEE Std 1003.1-2001, Chapter 2, Shell Command Language and other POSIX  
 3541 standards. Implementations may add symbols to the headers shown in the following table,  
 3542 provided the identifiers ... do not use the reserved prefixes `posix_`, `POSIX_`, or `_POSIX_`.”

3543 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/3 is applied, correcting the reserved  
 3544 macro prefix from: “`PRI[a-z]`, `SCN[a-z]`” to: “`PRI[Xa-z]`, `SCN[Xa-z]`” in the second table. The  
 3545 change was needed since the ISO C standard allows implementations to define macros of the  
 3546 form `PRI` or `SCN` followed by any lowercase letter or ‘`X`’ in `<inttypes.h>`. (The  
 3547 ISO / IEC 9899:1999 standard, Subclause 7.26.4.)

3548 IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/4 is applied, adding a new section listing  
 3549 reserved names for the `<stdint.h>` header. This change is for alignment with the ISO C standard.

### 3550 B.2.3 Error Numbers

3551 It was the consensus of the standard developers that to allow the conformance document to  
 3552 state that an error occurs and under what conditions, but to disallow a statement that it never  
 3553 occurs, does not make sense. It could be implied by the current wording that this is allowed, but  
 3554 to reduce the possibility of future interpretation requests, it is better to make an explicit  
 3555 statement.

3556 The ISO C standard requires that `errno` be an assignable lvalue. Originally, the definition in  
 3557 POSIX.1 was stricter than that in the ISO C standard, **extern int `errno`**, in order to support  
 3558 historical usage. In a multi-threaded environment, implementing `errno` as a global variable  
 3559 results in non-deterministic results when accessed. It is required, however, that `errno` work as a  
 3560 per-thread error reporting mechanism. In order to do this, a separate `errno` value has to be  
 3561 maintained for each thread. The following section discusses the various alternative solutions  
 3562 that were considered.

3563 In order to avoid this problem altogether for new functions, these functions avoid using `errno`  
 3564 and, instead, return the error number directly as the function return value; a return value of zero  
 3565 indicates that no error was detected.

3566 For any function that can return errors, the function return value is not used for any purpose  
 3567 other than for reporting errors. Even when the output of the function is scalar, it is passed  
 3568 through a function argument. While it might have been possible to allow some scalar outputs to  
 3569 be coded as negative function return values and mixed in with positive error status returns, this  
 3570 was rejected—using the return value for a mixed purpose was judged to be of limited use and  
 3571 error prone.

3572 Checking the value of *errno* alone is not sufficient to determine the existence or type of an error,  
 3573 since it is not required that a successful function call clear *errno*. The variable *errno* should only  
 3574 be examined when the return value of a function indicates that the value of *errno* is meaningful.  
 3575 In that case, the function is required to set the variable to something other than zero.

3576 The variable *errno* is never set to zero by any function call; to do so would contradict the ISO C  
 3577 standard.

3578 POSIX.1 requires (in the ERRORS sections of function descriptions) certain error values to be set  
 3579 in certain conditions because many existing applications depend on them. Some error numbers,  
 3580 such as [EFAULT], are entirely implementation-defined and are noted as such in their  
 3581 description in the ERRORS section. This section otherwise allows wide latitude to the  
 3582 implementation in handling error reporting.

3583 Some of the ERRORS sections in IEEE Std 1003.1-2001 have two subsections. The first:

3584 “The function shall fail if:”

3585 could be called the “mandatory” section.

3586 The second:

3587 “The function may fail if:”

3588 could be informally known as the “optional” section.

3589 Attempting to infer the quality of an implementation based on whether it detects optional error  
 3590 conditions is not useful.

3591 Following each one-word symbolic name for an error, there is a description of the error. The  
 3592 rationale for some of the symbolic names follows:

3593 [ECANCELED] This spelling was chosen as being more common.

3594 [EFAULT] Most historical implementations do not catch an error and set *errno* when an  
 3595 invalid address is given to the functions *wait()*, *time()*, or *times()*. Some  
 3596 implementations cannot reliably detect an invalid address. And most systems  
 3597 that detect invalid addresses will do so only for a system call, not for a library  
 3598 routine.

3599 [EFTYPE] This error code was proposed in earlier proposals as “Inappropriate operation  
 3600 for file type”, meaning that the operation requested is not appropriate for the  
 3601 file specified in the function call. This code was proposed, although the same  
 3602 idea was covered by [ENOTTY], because the connotations of the name would  
 3603 be misleading. It was pointed out that the *fcntl()* function uses the error code  
 3604 [EINVAL] for this notion, and hence all instances of [EFTYPE] were changed  
 3605 to this code.

3606 [EINTR] POSIX.1 prohibits conforming implementations from restarting interrupted  
 3607 system calls of conforming applications unless the SA\_RESTART flag is in  
 3608 effect for the signal. However, it does not require that [EINTR] be returned  
 3609 when another legitimate value may be substituted; for example, a partial  
 3610 transfer count when *read()* or *write()* are interrupted. This is only given when

|      |                |                                                                                                                                                                                                                |
|------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3611 |                | the signal-catching function returns normally as opposed to returns by                                                                                                                                         |
| 3612 |                | mechanisms like <i>longjmp()</i> or <i>siglongjmp()</i> .                                                                                                                                                      |
| 3613 | [ELOOP]        | In specifying conditions under which implementations would generate this                                                                                                                                       |
| 3614 |                | error, the following goals were considered:                                                                                                                                                                    |
| 3615 |                | <ul style="list-style-type: none"> <li>• To ensure that actual loops are detected, including loops that result from symbolic links across distributed file systems.</li> </ul>                                 |
| 3616 |                |                                                                                                                                                                                                                |
| 3617 |                | <ul style="list-style-type: none"> <li>• To ensure that during pathname resolution an application can rely on the ability to follow at least {SYMLOOP_MAX} symbolic links in the absence of a loop.</li> </ul> |
| 3618 |                |                                                                                                                                                                                                                |
| 3619 |                |                                                                                                                                                                                                                |
| 3620 |                | <ul style="list-style-type: none"> <li>• To allow implementations to provide the capability of traversing more than {SYMLOOP_MAX} symbolic links in the absence of a loop.</li> </ul>                          |
| 3621 |                |                                                                                                                                                                                                                |
| 3622 |                | <ul style="list-style-type: none"> <li>• To allow implementations to detect loops and generate the error prior to encountering {SYMLOOP_MAX} symbolic links.</li> </ul>                                        |
| 3623 |                |                                                                                                                                                                                                                |
| 3624 | [ENAMETOOLONG] |                                                                                                                                                                                                                |
| 3625 |                | When a symbolic link is encountered during pathname resolution, the                                                                                                                                            |
| 3626 |                | contents of that symbolic link are used to create a new pathname. The                                                                                                                                          |
| 3627 |                | standard developers intended to allow, but not require, that implementations                                                                                                                                   |
| 3628 |                | enforce the restriction of {PATH_MAX} on the result of this pathname                                                                                                                                           |
| 3629 |                | substitution.                                                                                                                                                                                                  |
| 3630 | [ENOMEM]       | The term “main memory” is not used in POSIX.1 because it is                                                                                                                                                    |
| 3631 |                | implementation-defined.                                                                                                                                                                                        |
| 3632 | [ENOTSUP]      | This error code is to be used when an implementation chooses to implement                                                                                                                                      |
| 3633 |                | the required functionality of IEEE Std 1003.1-2001 but does not support                                                                                                                                        |
| 3634 |                | optional facilities defined by IEEE Std 1003.1-2001. The return of [ENOSYS] is                                                                                                                                 |
| 3635 |                | to be taken to indicate that the function of the interface is not supported at all;                                                                                                                            |
| 3636 |                | the function will always fail with this error code.                                                                                                                                                            |
| 3637 | [ENOTTY]       | The symbolic name for this error is derived from a time when device control                                                                                                                                    |
| 3638 |                | was done by <i>ioctl()</i> and that operation was only permitted on a terminal                                                                                                                                 |
| 3639 |                | interface. The term “TTY” is derived from “teletypewriter”, the devices to                                                                                                                                     |
| 3640 |                | which this error originally applied.                                                                                                                                                                           |
| 3641 | [EOVERFLOW]    | Most of the uses of this error code are related to large file support. Typically,                                                                                                                              |
| 3642 |                | these cases occur on systems which support multiple programming                                                                                                                                                |
| 3643 |                | environments with different sizes for <b>off_t</b> , but they may also occur in                                                                                                                                |
| 3644 |                | connection with remote file systems.                                                                                                                                                                           |
| 3645 |                | In addition, when different programming environments have different widths                                                                                                                                     |
| 3646 |                | for types such as <b>int</b> and <b>uid_t</b> , several functions may encounter a condition                                                                                                                    |
| 3647 |                | where a value in a particular environment is too wide to be represented. In                                                                                                                                    |
| 3648 |                | that case, this error should be raised. For example, suppose the currently                                                                                                                                     |
| 3649 |                | running process has 64-bit <b>int</b> , and file descriptor 9 223 372 036 854 775 807 is                                                                                                                       |
| 3650 |                | open and does not have the close-on-exec flag set. If the process then uses                                                                                                                                    |
| 3651 |                | <i>execl()</i> to <i>exec</i> a file compiled in a programming environment with 32-bit <b>int</b> ,                                                                                                            |
| 3652 |                | the call to <i>execl()</i> can fail with <i>errno</i> set to [EOVERFLOW]. A similar failure                                                                                                                    |
| 3653 |                | can occur with <i>execl()</i> if any of the user IDs or any of the group IDs to be                                                                                                                             |
| 3654 |                | assigned to the new process image are out of range for the executed file's                                                                                                                                     |
| 3655 |                | programming environment.                                                                                                                                                                                       |

3656 Note, however, that this condition cannot occur for functions that are  
3657 explicitly described as always being successful, such as *getpid()*.

3658 [EPIPE] This condition normally generates the signal SIGPIPE; the error is returned if  
3659 the signal does not terminate the process.

3660 [EROFS] In historical implementations, attempting to *unlink()* or *rmdir()* a mount point  
3661 would generate an [EBUSY] error. An implementation could be envisioned  
3662 where such an operation could be performed without error. In this case, if  
3663 *either* the directory entry or the actual data structures reside on a read-only file  
3664 system, [EROFS] is the appropriate error to generate. (For example, changing  
3665 the link count of a file on a read-only file system could not be done, as is  
3666 required by *unlink()*, and thus an error should be reported.)

3667 Three error numbers, [EDOM], [EILSEQ], and [ERANGE], were added to this section primarily  
3668 for consistency with the ISO C standard.

### 3669 **Alternative Solutions for Per-Thread *errno***

3670 The usual implementation of *errno* as a single global variable does not work in a multi-threaded  
3671 environment. In such an environment, a thread may make a POSIX.1 call and get a *-1* error  
3672 return, but before that thread can check the value of *errno*, another thread might have made a  
3673 second POSIX.1 call that also set *errno*. This behavior is unacceptable in robust programs. There  
3674 were a number of alternatives that were considered for handling the *errno* problem:

- 3675 • Implement *errno* as a per-thread integer variable.
- 3676 • Implement *errno* as a service that can access the per-thread error number.
- 3677 • Change all POSIX.1 calls to accept an extra status argument and avoid setting *errno*.
- 3678 • Change all POSIX.1 calls to raise a language exception.

3679 The first option offers the highest level of compatibility with existing practice but requires  
3680 special support in the linker, compiler, and/or virtual memory system to support the new  
3681 concept of thread private variables. When compared with current practice, the third and fourth  
3682 options are much cleaner, more efficient, and encourage a more robust programming style, but  
3683 they require new versions of all of the POSIX.1 functions that might detect an error. The second  
3684 option offers compatibility with existing code that uses the **<errno.h>** header to define the  
3685 symbol *errno*. In this option, *errno* may be a macro defined:

```
3686 #define errno (*__errno())
3687 extern int *__errno();
```

3688 This option may be implemented as a per-thread variable whereby an *errno* field is allocated in  
3689 the user space object representing a thread, and whereby the function *\_\_errno()* makes a system  
3690 call to determine the location of its user space object and returns the address of the *errno* field of  
3691 that object. Another implementation, one that avoids calling the kernel, involves allocating  
3692 stacks in chunks. The stack allocator keeps a side table indexed by chunk number containing a  
3693 pointer to the thread object that uses that chunk. The *\_\_errno()* function then looks at the stack  
3694 pointer, determines the chunk number, and uses that as an index into the chunk table to find its  
3695 thread object and thus its private value of *errno*. On most architectures, this can be done in four  
3696 to five instructions. Some compilers may wish to implement *\_\_errno()* inline to improve  
3697 performance.

### Disallowing Return of the [EINTR] Error Code

Many blocking interfaces defined by IEEE Std 1003.1-2001 may return [EINTR] if interrupted during their execution by a signal handler. Blocking interfaces introduced under the Threads option do not have this property. Instead, they require that the interface appear to be atomic with respect to interruption. In particular, clients of blocking interfaces need not handle any possible [EINTR] return as a special case since it will never occur. If it is necessary to restart operations or complete incomplete operations following the execution of a signal handler, this is handled by the implementation, rather than by the application.

Requiring applications to handle [EINTR] errors on blocking interfaces has been shown to be a frequent source of often unreproducible bugs, and it adds no compelling value to the available functionality. Thus, blocking interfaces introduced for use by multi-threaded programs do not use this paradigm. In particular, in none of the functions *flockfile()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*, *pthread\_join()*, *pthread\_mutex\_lock()*, and *sigwait()* did providing [EINTR] returns add value, or even particularly make sense. Thus, these functions do not provide for an [EINTR] return, even when interrupted by a signal handler. The same arguments can be applied to *sem\_wait()*, *sem\_trywait()*, *sigwaitinfo()*, and *sigtimedwait()*, but implementations are permitted to return [EINTR] error codes for these functions for compatibility with earlier versions of IEEE Std 1003.1. Applications cannot rely on calls to these functions returning [EINTR] error codes when signals are delivered to the calling thread, but they should allow for the possibility.

#### B.2.3.1 Additional Error Numbers

The ISO C standard defines the name space for implementations to add additional error numbers.

### B.2.4 Signal Concepts

Historical implementations of signals, using the *signal()* function, have shortcomings that make them unreliable for many application uses. Because of this, a new signal mechanism, based very closely on the one of 4.2 BSD and 4.3 BSD, was added to POSIX.1.

#### Signal Names

The restriction on the actual type used for **sigset\_t** is intended to guarantee that these objects can always be assigned, have their address taken, and be passed as parameters by value. It is not intended that this type be a structure including pointers to other data structures, as that could impact the portability of applications performing such operations. A reasonable implementation could be a structure containing an array of some integer type.

The signals described in IEEE Std 1003.1-2001 must have unique values so that they may be named as parameters of **case** statements in the body of a C-language **switch** clause. However, implementation-defined signals may have values that overlap with each other or with signals specified in IEEE Std 1003.1-2001. An example of this is SIGABRT, which traditionally overlaps some other signal, such as SIGIOT.

SIGKILL, SIGTERM, SIGUSR1, and SIGUSR2 are ordinarily generated only through the explicit use of the *kill()* function, although some implementations generate SIGKILL under extraordinary circumstances. SIGTERM is traditionally the default signal sent by the *kill* command.

The signals SIGBUS, SIGEMT, SIGIOT, SIGTRAP, and SIGSYS were omitted from POSIX.1 because their behavior is implementation-defined and could not be adequately categorized. Conforming implementations may deliver these signals, but must document the circumstances

under which they are delivered and note any restrictions concerning their delivery. The signals SIGFPE, SIGILL, and SIGSEGV are similar in that they also generally result only from programming errors. They were included in POSIX.1 because they do indicate three relatively well-categorized conditions. They are all defined by the ISO C standard and thus would have to be defined by any system with an ISO C standard binding, even if not explicitly included in POSIX.1.

There is very little that a Conforming POSIX.1 Application can do by catching, ignoring, or masking any of the signals SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGBUS, SIGSEGV, SIGSYS, or SIGFPE. They will generally be generated by the system only in cases of programming errors. While it may be desirable for some robust code (for example, a library routine) to be able to detect and recover from programming errors in other code, these signals are not nearly sufficient for that purpose. One portable use that does exist for these signals is that a command interpreter can recognize them as the cause of a process' termination (with *wait()*) and print an appropriate message. The mnemonic tags for these signals are derived from their PDP-11 origin.

The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are provided for job control and are unchanged from 4.2 BSD. The signal SIGCHLD is also typically used by job control shells to detect children that have terminated or, as in 4.2 BSD, stopped.

Some implementations, including System V, have a signal named SIGCLD, which is similar to SIGCHLD in 4.2 BSD. POSIX.1 permits implementations to have a single signal with both names. POSIX.1 carefully specifies ways in which conforming applications can avoid the semantic differences between the two different implementations. The name SIGCHLD was chosen for POSIX.1 because most current application usages of it can remain unchanged in conforming applications. SIGCLD in System V has more cases of semantics that POSIX.1 does not specify, and thus applications using it are more likely to require changes in addition to the name change.

The signals SIGUSR1 and SIGUSR2 are commonly used by applications for notification of exceptional behavior and are described as “reserved as application-defined” so that such use is not prohibited. Implementations should not generate SIGUSR1 or SIGUSR2, except when explicitly requested by *kill()*. It is recommended that libraries not use these two signals, as such use in libraries could interfere with their use by applications calling the libraries. If such use is unavoidable, it should be documented. It is prudent for non-portable libraries to use non-standard signals to avoid conflicts with use of standard signals by portable libraries.

There is no portable way for an application to catch or ignore non-standard signals. Some implementations define the range of signal numbers, so applications can install signal-catching functions for all of them. Unfortunately, implementation-defined signals often cause problems when caught or ignored by applications that do not understand the reason for the signal. While the desire exists for an application to be more robust by handling all possible signals (even those only generated by *kill()*), no existing mechanism was found to be sufficiently portable to include in POSIX.1. The value of such a mechanism, if included, would be diminished given that SIGKILL would still not be catchable.

A number of new signal numbers are reserved for applications because the two user signals defined by POSIX.1 are insufficient for many realtime applications. A range of signal numbers is specified, rather than an enumeration of additional reserved signal names, because different applications and application profiles will require a different number of application signals. It is not desirable to burden all application domains and therefore all implementations with the maximum number of signals required by all possible applications. Note that in this context, signal numbers are essentially different signal priorities.

The relatively small number of required additional signals, {\_POSIX\_RTSIG\_MAX}, was chosen so as not to require an unreasonably large signal mask/set. While this number of signals defined

in POSIX.1 will fit in a single 32-bit word signal mask, it is recognized that most existing implementations define many more signals than are specified in POSIX.1 and, in fact, many implementations have already exceeded 32 signals (including the “null signal”). Support of `{_POSIX_RTSIG_MAX}` additional signals may push some implementation over the single 32-bit word line, but is unlikely to push any implementations that are already over that line beyond the 64-signal line.

#### B.2.4.1 Signal Generation and Delivery

The terms defined in this section are not used consistently in documentation of historical systems. Each signal can be considered to have a lifetime beginning with generation and ending with delivery or acceptance. The POSIX.1 definition of “delivery” does not exclude ignored signals; this is considered a more consistent definition. This revised text in several parts of IEEE Std 1003.1-2001 clarifies the distinct semantics of asynchronous signal delivery and synchronous signal acceptance. The previous wording attempted to categorize both under the term “delivery”, which led to conflicts over whether the effects of asynchronous signal delivery applied to synchronous signal acceptance.

Signals generated for a process are delivered to only one thread. Thus, if more than one thread is eligible to receive a signal, one has to be chosen. The choice of threads is left entirely up to the implementation both to allow the widest possible range of conforming implementations and to give implementations the freedom to deliver the signal to the “easiest possible” thread should there be differences in ease of delivery between different threads.

Note that should multiple delivery among cooperating threads be required by an application, this can be trivially constructed out of the provided single-delivery semantics. The construction of a `sigwait_multiple()` function that accomplishes this goal is presented with the rationale for `sigwaitinfo()`.

Implementations should deliver unblocked signals as soon after they are generated as possible. However, it is difficult for POSIX.1 to make specific requirements about this, beyond those in `kill()` and `sigprocmask()`. Even on systems with prompt delivery, scheduling of higher priority processes is always likely to cause delays.

In general, the interval between the generation and delivery of unblocked signals cannot be detected by an application. Thus, references to pending signals generally apply to blocked, pending signals. An implementation registers a signal as pending on the process when no thread has the signal unblocked and there are no threads blocked in a `sigwait()` function for that signal. Thereafter, the implementation delivers the signal to the first thread that unblocks the signal or calls a `sigwait()` function on a signal set containing this signal rather than choosing the recipient thread at the time the signal is sent.

In the 4.3 BSD system, signals that are blocked and set to `SIG_IGN` are discarded immediately upon generation. For a signal that is ignored as its default action, if the action is `SIG_DFL` and the signal is blocked, a generated signal remains pending. In the 4.1 BSD system and in System V Release 3 (two other implementations that support a somewhat similar signal mechanism), all ignored blocked signals remain pending if generated. Because it is not normally useful for an application to simultaneously ignore and block the same signal, it was unnecessary for POSIX.1 to specify behavior that would invalidate any of the historical implementations.

There is one case in some historical implementations where an unblocked, pending signal does not remain pending until it is delivered. In the System V implementation of `signal()`, pending signals are discarded when the action is set to `SIG_DFL` or a signal-catching routine (as well as to `SIG_IGN`). Except in the case of setting `SIGCHLD` to `SIG_DFL`, implementations that do this do not conform completely to POSIX.1. Some earlier proposals for POSIX.1 explicitly stated this, but these statements were redundant due to the requirement that functions defined by POSIX.1

not change attributes of processes defined by POSIX.1 except as explicitly stated.

POSIX.1 specifically states that the order in which multiple, simultaneously pending signals are delivered is unspecified. This order has not been explicitly specified in historical implementations, but has remained quite consistent and been known to those familiar with the implementations. Thus, there have been cases where applications (usually system utilities) have been written with explicit or implicit dependencies on this order. Implementors and others porting existing applications may need to be aware of such dependencies.

When there are multiple pending signals that are not blocked, implementations should arrange for the delivery of all signals at once, if possible. Some implementations stack calls to all pending signal-catching routines, making it appear that each signal-catcher was interrupted by the next signal. In this case, the implementation should ensure that this stacking of signals does not violate the semantics of the signal masks established by *sigaction()*. Other implementations process at most one signal when the operating system is entered, with remaining signals saved for later delivery. Although this practice is widespread, this behavior is neither standardized nor endorsed. In either case, implementations should attempt to deliver signals associated with the current state of the process (for example, SIGFPE) before other signals, if possible.

In 4.2 BSD and 4.3 BSD, it is not permissible to ignore or explicitly block SIGCONT, because if blocking or ignoring this signal prevented it from continuing a stopped process, such a process could never be continued (only killed by SIGKILL). However, 4.2 BSD and 4.3 BSD do block SIGCONT during execution of its signal-catching function when it is caught, creating exactly this problem. A proposal was considered to disallow catching SIGCONT in addition to ignoring and blocking it, but this limitation led to objections. The consensus was to require that SIGCONT always continue a stopped process when generated. This removed the need to disallow ignoring or explicit blocking of the signal; note that SIG\_IGN and SIG\_DFL are equivalent for SIGCONT.

#### B.2.4.2 Realtime Signal Generation and Delivery

The Realtime Signals Extension option to POSIX.1 signal generation and delivery behavior is required for the following reasons:

- The **sigevent** structure is used by other POSIX.1 functions that result in asynchronous event notifications to specify the notification mechanism to use and other information needed by the notification mechanism. IEEE Std 1003.1-2001 defines only three symbolic values for the notification mechanism:

- SIGEV\_NONE is used to indicate that no notification is required when the event occurs. This is useful for applications that use asynchronous I/O with polling for completion.

- SIGEV\_SIGNAL indicates that a signal is generated when the event occurs.

- SIGEV\_THREAD provides for “callback functions” for asynchronous notifications done by a function call within the context of a new thread. This provides a multi-threaded process with a more natural means of notification than signals.

The primary difficulty with previous notification approaches has been to specify the environment of the notification routine.

- One approach is to limit the notification routine to call only functions permitted in a signal handler. While the list of permissible functions is clearly stated, this is overly restrictive.

- A second approach is to define a new list of functions or classes of functions that are explicitly permitted or not permitted. This would give a programmer more lists to deal with, which would be awkward.



— The third approach is to define completely the environment for execution of the notification function. A clear definition of an execution environment for notification is provided by executing the notification function in the environment of a newly created thread.

Implementations may support additional notification mechanisms by defining new values for *sigev\_notify*.

For a notification type of `SIGEV_SIGNAL`, the other members of the **sigevent** structure defined by IEEE Std 1003.1-2001 specify the realtime signal—that is, the signal number and application-defined value that differentiates between occurrences of signals with the same number—that will be generated when the event occurs. The structure is defined in `<signal.h>`, even though the structure is not directly used by any of the signal functions, because it is part of the signals interface used by the POSIX.1b “client functions”. When the client functions include `<signal.h>` to define the signal names, the **sigevent** structure will also be defined.

An application-defined value passed to the signal handler is used to differentiate between different “events” instead of requiring that the application use different signal numbers for several reasons:

— Realtime applications potentially handle a very large number of different events. Requiring that implementations support a correspondingly large number of distinct signal numbers will adversely impact the performance of signal delivery because the signal masks to be manipulated on entry and exit to the handlers will become large.

— Event notifications are prioritized by signal number (the rationale for this is explained in the following paragraphs) and the use of different signal numbers to differentiate between the different event notifications overloads the signal number more than has already been done. It also requires that the application writer make arbitrary assignments of priority to events that are logically of equal priority.

A union is defined for the application-defined value so that either an integer constant or a pointer can be portably passed to the signal-catching function. On some architectures a pointer cannot be cast to an **int** and *vice versa*.

Use of a structure here with an explicit notification type discriminant rather than explicit parameters to realtime functions, or embedded in other realtime structures, provides for future extensions to IEEE Std 1003.1-2001. Additional, perhaps more efficient, notification mechanisms can be supported for existing realtime function interfaces, such as timers and asynchronous I/O, by extending the **sigevent** structure appropriately. The existing realtime function interfaces will not have to be modified to use any such new notification mechanism. The revised text concerning the `SIGEV_SIGNAL` value makes consistent the semantics of the members of the **sigevent** structure, particularly in the definitions of *lio\_listio()* and *aio\_fsync()*. For uniformity, other revisions cause this specification to be referred to rather than inaccurately duplicated in the descriptions of functions and structures using the **sigevent** structure. The revised wording does not relax the requirement that the signal number be in the range `SIGRTMIN` to `SIGRTMAX` to guarantee queuing and passing of the application value, since that requirement is still implied by the signal names.

- IEEE Std 1003.1-2001 is intentionally vague on whether “non-realtime” signal-generating mechanisms can result in a **siginfo\_t** being supplied to the handler on delivery. In one existing implementation, a **siginfo\_t** is posted on signal generation, even though the implementation does not support queuing of multiple occurrences of a signal. It is not the intent of IEEE Std 1003.1-2001 to preclude this, independent of the mandate to define signals that do support queuing. Any interpretation that appears to preclude this is a mistake in the

reading or writing of the standard.

- Signals handled by realtime signal handlers might be generated by functions or conditions that do not allow the specification of an application-defined value and do not queue. IEEE Std 1003.1-2001 specifies the *si\_code* member of the **siginfo\_t** structure used in existing practice and defines additional codes so that applications can detect whether an application-defined value is present or not. The code `SI_USER` for `kill()`-generated signals is adopted from existing practice.

- The `sigaction()` *sa\_flags* value `SA_SIGINFO` tells the implementation that the signal-catching function expects two additional arguments. When the flag is not set, a single argument, the signal number, is passed as specified by IEEE Std 1003.1-2001. Although IEEE Std 1003.1-2001 does not explicitly allow the *info* argument to the handler function to be `NULL`, this is existing practice. This provides for compatibility with programs whose signal-catching functions are not prepared to accept the additional arguments. IEEE Std 1003.1-2001 is explicitly unspecified as to whether signals actually queue when `SA_SIGINFO` is not set for a signal, as there appear to be no benefits to applications in specifying one behavior or another. One existing implementation queues a **siginfo\_t** on each signal generation, unless the signal is already pending, in which case the implementation discards the new **siginfo\_t**; that is, the queue length is never greater than one. This implementation only examines `SA_SIGINFO` on signal delivery, discarding the queued **siginfo\_t** if its delivery was not requested.

IEEE Std 1003.1-2001 specifies several new values for the *si\_code* member of the **siginfo\_t** structure. In existing practice, a *si\_code* value of less than or equal to zero indicates that the signal was generated by a process via the `kill()` function. In existing practice, values of *si\_code* that provide additional information for implementation-generated signals, such as `SIGFPE` or `SIGSEGV`, are all positive. Thus, if implementations define the new constants specified in IEEE Std 1003.1-2001 to be negative numbers, programs written to use existing practice will not break. IEEE Std 1003.1-2001 chose not to attempt to specify existing practice values of *si\_code* other than `SI_USER` both because it was deemed beyond the scope of IEEE Std 1003.1-2001 and because many of the values in existing practice appear to be platform and implementation-defined. But, IEEE Std 1003.1-2001 does specify that if an implementation—for example, one that does not have existing practice in this area—chooses to define additional values for *si\_code*, these values have to be different from the values of the symbols specified by IEEE Std 1003.1-2001. This will allow conforming applications to differentiate between signals generated by one of the POSIX.1b asynchronous events and those generated by other implementation events in a manner compatible with existing practice.

The unique values of *si\_code* for the POSIX.1b asynchronous events have implications for implementations of, for example, asynchronous I/O or message passing in user space library code. Such an implementation will be required to provide a hidden interface to the signal generation mechanism that allows the library to specify the standard values of *si\_code*.

Existing practice also defines additional members of **siginfo\_t**, such as the process ID and user ID of the sending process for `kill()`-generated signals. These members were deemed not necessary to meet the requirements of realtime applications and are not specified by IEEE Std 1003.1-2001. Neither are they precluded.

The third argument to the signal-catching function, *context*, is left undefined by IEEE Std 1003.1-2001, but is specified in the interface because it matches existing practice for the `SA_SIGINFO` flag. It was considered undesirable to require a separate implementation for `SA_SIGINFO` for POSIX conformance on implementations that already support the two additional parameters.

- The requirement to deliver lower numbered signals in the range SIGRTMIN to SIGRTMAX first, when multiple unblocked signals are pending, results from several considerations:

- A method is required to prioritize event notifications. The signal number was chosen instead of, for instance, associating a separate priority with each request, because an implementation has to check pending signals at various points and select one for delivery when more than one is pending. Specifying a selection order is the minimal additional semantic that will achieve prioritized delivery. If a separate priority were to be associated with queued signals, it would be necessary for an implementation to search all non-empty, non-blocked signal queues and select from among them the pending signal with the highest priority. This would significantly increase the cost of and decrease the determinism of signal delivery.

- Given the specified selection of the lowest numeric unblocked pending signal, preemptive priority signal delivery can be achieved using signal numbers and signal masks by ensuring that the *sa\_mask* for each signal number blocks all signals with a higher numeric value.

For realtime applications that want to use only the newly defined realtime signal numbers without interference from the standard signals, this can be achieved by blocking all of the standard signals in the thread signal mask and in the *sa\_mask* installed by the signal action for the realtime signal handlers.

IEEE Std 1003.1-2001 explicitly leaves unspecified the ordering of signals outside of the range of realtime signals and the ordering of signals within this range with respect to those outside the range. It was believed that this would unduly constrain implementations or standards in the future definition of new signals.

### B.2.4.3 Signal Actions

Early proposals mentioned SIGCONT as a second exception to the rule that signals are not delivered to stopped processes until continued. Because IEEE Std 1003.1-2001 now specifies that SIGCONT causes the stopped process to continue when it is generated, delivery of SIGCONT is not prevented because a process is stopped, even without an explicit exception to this rule.

Ignoring a signal by setting the action to SIG\_IGN (or SIG\_DFL for signals whose default action is to ignore) is not the same as installing a signal-catching function that simply returns. Invoking such a function will interrupt certain system functions that block processes (for example, *wait()*, *sigsuspend()*, *pause()*, *read()*, *write()*) while ignoring a signal has no such effect on the process.

Historical implementations discard pending signals when the action is set to SIG\_IGN. However, they do not always do the same when the action is set to SIG\_DFL and the default action is to ignore the signal. IEEE Std 1003.1-2001 requires this for the sake of consistency and also for completeness, since the only signal this applies to is SIGCHLD, and IEEE Std 1003.1-2001 disallows setting its action to SIG\_IGN.

Some implementations (System V, for example) assign different semantics for SIGCLD depending on whether the action is set to SIG\_IGN or SIG\_DFL. Since POSIX.1 requires that the default action for SIGCHLD be to ignore the signal, applications should always set the action to SIG\_DFL in order to avoid SIGCHLD.

Whether or not an implementation allows SIG\_IGN as a SIGCHLD disposition to be inherited across a call to one of the *exec* family of functions or *posix\_spawn()* is explicitly left as unspecified. This change was made as a result of IEEE PASC Interpretation 1003.1 #132, and permits the implementation to decide between the following alternatives:

- Unconditionally leave SIGCHLD set to SIG\_IGN, in which case the implementation would not allow applications that assume inheritance of SIG\_DFL to conform to IEEE Std 1003.1-2001 without change. The implementation would, however, retain an ability to control applications that create child processes but never call on the *wait* family of functions, potentially filling up the process table.
- Unconditionally reset SIGCHLD to SIG\_DFL, in which case the implementation would allow applications that assume inheritance of SIG\_DFL to conform. The implementation would, however, lose an ability to control applications that spawn child processes but never reap them.
- Provide some mechanism, not specified in IEEE Std 1003.1-2001, to control inherited SIGCHLD dispositions.

Some implementations (System V, for example) will deliver a SIGCHLD signal immediately when a process establishes a signal-catching function for SIGCHLD when that process has a child that has already terminated. Other implementations, such as 4.3 BSD, do not generate a new SIGCHLD signal in this way. In general, a process should not attempt to alter the signal action for the SIGCHLD signal while it has any outstanding children. However, it is not always possible for a process to avoid this; for example, shells sometimes start up processes in pipelines with other processes from the pipeline as children. Processes that cannot ensure that they have no children when altering the signal action for SIGCHLD thus need to be prepared for, but not depend on, generation of an immediate SIGCHLD signal.

The default action of the stop signals (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is to stop a process that is executing. If a stop signal is delivered to a process that is already stopped, it has no effect. In fact, if a stop signal is generated for a stopped process whose signal mask blocks the signal, the signal will never be delivered to the process since the process must receive a SIGCONT, which discards all pending stop signals, in order to continue executing.

The SIGCONT signal continues a stopped process even if SIGCONT is blocked (or ignored). However, if a signal-catching routine has been established for SIGCONT, it will not be entered until SIGCONT is unblocked.

If a process in an orphaned process group stops, it is no longer under the control of a job control shell and hence would not normally ever be continued. Because of this, orphaned processes that receive terminal-related stop signals (SIGTSTP, SIGTTIN, SIGTTOU, but not SIGSTOP) must not be allowed to stop. The goal is to prevent stopped processes from languishing forever. (As SIGSTOP is sent only via *kill()*, it is assumed that the process or user sending a SIGSTOP can send a SIGCONT when desired.) Instead, the system must discard the stop signal. As an extension, it may also deliver another signal in its place. 4.3 BSD sends a SIGKILL, which is overly effective because SIGKILL is not catchable. Another possible choice is SIGHUP. 4.3 BSD also does this for orphaned processes (processes whose parent has terminated) rather than for members of orphaned process groups; this is less desirable because job control shells manage process groups. POSIX.1 also prevents SIGTTIN and SIGTTOU signals from being generated for processes in orphaned process groups as a direct result of activity on a terminal, preventing infinite loops when *read()* and *write()* calls generate signals that are discarded; see Section A.11.1.4 (on page 67). A similar restriction on the generation of SIGTSTP was considered, but that would be unnecessary and more difficult to implement due to its asynchronous nature.

Although POSIX.1 requires that signal-catching functions be called with only one argument, there is nothing to prevent conforming implementations from extending POSIX.1 to pass additional arguments, as long as Strictly Conforming POSIX.1 Applications continue to compile and execute correctly. Most historical implementations do, in fact, pass additional, signal-specific arguments to certain signal-catching routines.

There was a proposal to change the declared type of the signal handler to:

```
void func (int sig, ...);
```

The usage of ellipses ("...") is ISO C standard syntax to indicate a variable number of arguments. Its use was intended to allow the implementation to pass additional information to the signal handler in a standard manner.

Unfortunately, this construct would require all signal handlers to be defined with this syntax because the ISO C standard allows implementations to use a different parameter passing mechanism for variable parameter lists than for non-variable parameter lists. Thus, all existing signal handlers in all existing applications would have to be changed to use the variable syntax in order to be standard and portable. This is in conflict with the goal of Minimal Changes to Existing Application Code.

When terminating a process from a signal-catching function, processes should be aware of any interpretation that their parent may make of the status returned by *wait()* or *waitpid()*. In particular, a signal-catching function should not call *exit(0)* or *\_exit(0)* unless it wants to indicate successful termination. A non-zero argument to *exit()* or *\_exit()* can be used to indicate unsuccessful termination. Alternatively, the process can use *kill()* to send itself a fatal signal (first ensuring that the signal is set to the default action and not blocked). See also the RATIONALE section of the *\_exit()* function.

The behavior of *unsafe* functions, as defined by this section, is undefined when they are invoked from signal-catching functions in certain circumstances. The behavior of reentrant functions, as defined by this section, is as specified by POSIX.1, regardless of invocation from a signal-catching function. This is the only intended meaning of the statement that reentrant functions may be used in signal-catching functions without restriction. Applications must still consider all effects of such functions on such things as data structures, files, and process state. In particular, application writers need to consider the restrictions on interactions when interrupting *sleep()* (see *sleep()*) and interactions among multiple handles for a file description. The fact that any specific function is listed as reentrant does not necessarily mean that invocation of that function from a signal-catching function is recommended.

In order to prevent errors arising from interrupting non-reentrant function calls, applications should protect calls to these functions either by blocking the appropriate signals or through the use of some programmatic semaphore. POSIX.1 does not address the more general problem of synchronizing access to shared data structures. Note in particular that even the "safe" functions may modify the global variable *errno*; the signal-catching function may want to save and restore its value. The same principles apply to the reentrancy of application routines and asynchronous data access.

Note that *longjmp()* and *siglongjmp()* are not in the list of reentrant functions. This is because the code executing after *longjmp()* or *siglongjmp()* can call any unsafe functions with the same danger as calling those unsafe functions directly from the signal handler. Applications that use *longjmp()* or *siglongjmp()* out of signal handlers require rigorous protection in order to be portable. Many of the other functions that are excluded from the list are traditionally implemented using either the C language *malloc()* or *free()* functions or the ISO C standard I/O library, both of which traditionally use data structures in a non-reentrant manner. Because any combination of different functions using a common data structure can cause reentrancy problems, POSIX.1 does not define the behavior when any unsafe function is called in a signal handler that interrupts any unsafe function.

The only realtime extension to signal actions is the addition of the additional parameters to the signal-catching function. This extension has been explained and motivated in the previous section. In making this extension, though, developers of POSIX.1b ran into issues relating to

function prototypes. In response to input from the POSIX.1 standard developers, members were added to the **sigaction** structure to specify function prototypes for the newer signal-catching function specified by POSIX.1b. These members follow changes that are being made to POSIX.1. Note that IEEE Std 1003.1-2001 explicitly states that these fields may overlap so that a union can be defined. This enabled existing implementations of POSIX.1 to maintain binary-compatibility when these extensions were added.

The **siginfo\_t** structure was adopted for passing the application-defined value to match existing practice, but the existing practice has no provision for an application-defined value, so this was added. Note that POSIX normally reserves the “\_t” type designation for opaque types. The **siginfo\_t** structure breaks with this convention to follow existing practice and thus promote portability. Standardization of the existing practice for the other members of this structure may be addressed in the future.

Although it is not explicitly visible to applications, there are additional semantics for signal actions implied by queued signals and their interaction with other POSIX.1b realtime functions. Specifically:

- It is not necessary to queue signals whose action is SIG\_IGN.
- For implementations that support POSIX.1b timers, some interaction with the timer functions at signal delivery is implied to manage the timer overrun count.

IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/5 is applied, reordering the RTS shaded text under the third and fourth paragraphs of the SIG\_DFL description. This corrects an earlier editorial error in this section.

IEEE Std 1003.1-2001 / Cor 1-2002, item XSH/TC1/D6/6 is applied, adding the *abort()* function to the list of async-cancel-safe functions.

#### B.2.4.4 Signal Effects on Other Functions

The most common behavior of an interrupted function after a signal-catching function returns is for the interrupted function to give an [EINTR] error unless the SA\_RESTART flag is in effect for the signal. However, there are a number of specific exceptions, including *sleep()* and certain situations with *read()* and *write()*.

The historical implementations of many functions defined by IEEE Std 1003.1-2001 are not interruptible, but delay delivery of signals generated during their execution until after they complete. This is never a problem for functions that are guaranteed to complete in a short (imperceptible to a human) period of time. It is normally those functions that can suspend a process indefinitely or for long periods of time (for example, *wait()*, *pause()*, *sigsuspend()*, *sleep()*, or *read()/write()* on a slow device like a terminal) that are interruptible. This permits applications to respond to interactive signals or to set timeouts on calls to most such functions with *alarm()*. Therefore, implementations should generally make such functions (including ones defined as extensions) interruptible.

Functions not mentioned explicitly as interruptible may be so on some implementations, possibly as an extension where the function gives an [EINTR] error. There are several functions (for example, *getpid()*, *getuid()*) that are specified as never returning an error, which can thus never be extended in this way.

If a signal-catching function returns while the SA\_RESTART flag is in effect, an interrupted function is restarted at the point it was interrupted. Conforming applications cannot make assumptions about the internal behavior of interrupted functions, even if the functions are async-signal-safe. For example, suppose the *read()* function is interrupted with SA\_RESTART in effect, the signal-catching function closes the file descriptor being read from and returns, and the

4169 *read()* function is then restarted; in this case the application cannot assume that the *read()*  
4170 function will give an [EBADF] error, since *read()* might have checked the file descriptor for  
4171 validity before being interrupted.

## 4172 **B.2.5 Standard I/O Streams**

### 4173 *B.2.5.1 Interaction of File Descriptors and Standard I/O Streams*

4174 There is no additional rationale provided for this section.

### 4175 *B.2.5.2 Stream Orientation and Encoding Rules*

4176 There is no additional rationale provided for this section.

## 4177 **B.2.6 STREAMS**

4178 STREAMS are introduced into IEEE Std 1003.1-2001 as part of the alignment with the Single  
4179 UNIX Specification, but marked as an option in recognition that not all systems may wish to  
4180 implement the facility. The option within IEEE Std 1003.1-2001 is denoted by the XSR margin  
4181 marker. The standard developers made this option independent of the XSI option.

4182 STREAMS are a method of implementing network services and other character-based  
4183 input/output mechanisms, with the STREAM being a full-duplex connection between a process  
4184 and a device. STREAMS provides direct access to protocol modules, and optional protocol  
4185 modules can be interposed between the process-end of the STREAM and the device-driver at the  
4186 device-end of the STREAM. Pipes can be implemented using the STREAMS mechanism, so they  
4187 can provide process-to-process as well as process-to-device communications.

4188 This section introduces STREAMS I/O, the message types used to control them, an overview of  
4189 the priority mechanism, and the interfaces used to access them.

### 4190 *B.2.6.1 Accessing STREAMS*

4191 There is no additional rationale provided for this section.

## 4192 **B.2.7 XSI Interprocess Communication**

4193 There are two forms of IPC supported as options in IEEE Std 1003.1-2001. The traditional  
4194 System V IPC routines derived from the SVID—that is, the *msg\*()*, *sem\*()*, and *shm\*()*  
4195 interfaces—are mandatory on XSI-conformant systems. Thus, all XSI-conformant systems  
4196 provide the same mechanisms for manipulating messages, shared memory, and semaphores.

4197 In addition, the POSIX Realtime Extension provides an alternate set of routines for those systems  
4198 supporting the appropriate options.

4199 The application writer is presented with a choice: the System V interfaces or the POSIX  
4200 interfaces (loosely derived from the Berkeley interfaces). The XSI profile prefers the System V  
4201 interfaces, but the POSIX interfaces may be more suitable for realtime or other performance-  
4202 sensitive applications.

4203 **B.2.7.1** *IPC General Information*

4204 General information that is shared by all three mechanisms is described in this section. The  
 4205 common permissions mechanism is briefly introduced, describing the mode bits, and how they  
 4206 are used to determine whether or not a process has access to read or write/alter the appropriate  
 4207 instance of one of the IPC mechanisms. All other relevant information is contained in the  
 4208 reference pages themselves.

4209 The semaphore type of IPC allows processes to communicate through the exchange of  
 4210 semaphore values. A semaphore is a positive integer. Since many applications require the use of  
 4211 more than one semaphore, XSI-conformant systems have the ability to create sets or arrays of  
 4212 semaphores.

4213 Calls to support semaphores include:

4214 `semctl()`, `semget()`, `semop()`

4215 Semaphore sets are created by using the `semget()` function.

4216 The message type of IPC allows processes to communicate through the exchange of data stored  
 4217 in buffers. This data is transmitted between processes in discrete portions known as messages.

4218 Calls to support message queues include:

4219 `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()`

4220 The shared memory type of IPC allows two or more processes to share memory and  
 4221 consequently the data contained therein. This is done by allowing processes to set up access to a  
 4222 common memory address space. This sharing of memory provides a fast means of exchange of  
 4223 data between processes.

4224 Calls to support shared memory include:

4225 `shmctl()`, `shmdt()`, `shmget()`

4226 The `ftok()` interface is also provided.

4227 **B.2.8** **Realtime**4228 **Advisory Information**

4229 POSIX.1b contains an Informative Annex with proposed interfaces for “realtime files”. These  
 4230 interfaces could determine groups of the exact parameters required to do “direct I/O” or  
 4231 “extents”. These interfaces were objected to by a significant portion of the balloting group as too  
 4232 complex. A conforming application had little chance of correctly navigating the large parameter  
 4233 space to match its desires to the system. In addition, they only applied to a new type of file  
 4234 (realtime files) and they told the implementation exactly what to do as opposed to advising the  
 4235 implementation on application behavior and letting it optimize for the system the (portable)  
 4236 application was running on. For example, it was not clear how a system that had a disk array  
 4237 should set its parameters.

4238 There seemed to be several overall goals:

- 4239 • Optimizing sequential access
- 4240 • Optimizing caching behavior
- 4241 • Optimizing I/O data transfer
- 4242 • Preallocation



The advisory interfaces, *posix\_fadvise()* and *posix\_madvise()*, satisfy the first two goals. The `POSIX_FADV_SEQUENTIAL` and `POSIX_MADV_SEQUENTIAL` advice tells the implementation to expect serial access. Typically the system will prefetch the next several serial accesses in order to overlap I/O. It may also free previously accessed serial data if memory is tight. If the application is not doing serial access it can use `POSIX_FADV_WILLNEED` and `POSIX_MADV_WILLNEED` to accomplish I/O overlap, as required. When the application advises `POSIX_FADV_RANDOM` or `POSIX_MADV_RANDOM` behavior, the implementation usually tries to fetch a minimum amount of data with each request and it does not expect much locality. `POSIX_FADV_DONTNEED` and `POSIX_MADV_DONTNEED` allow the system to free up caching resources as the data will not be required in the near future.

`POSIX_FADV_NOREUSE` tells the system that caching the specified data is not optimal. For file I/O, the transfer should go directly to the user buffer instead of being cached internally by the implementation. To portably perform direct disk I/O on all systems, the application must perform its I/O transfers according to the following rules:

1. The user buffer should be aligned according to the `{POSIX_REC_XFER_ALIGN}` *pathconf()* variable.
2. The number of bytes transferred in an I/O operation should be a multiple of the `{POSIX_ALLOC_SIZE_MIN}` *pathconf()* variable.
3. The offset into the file at the start of an I/O operation should be a multiple of the `{POSIX_ALLOC_SIZE_MIN}` *pathconf()* variable.
4. The application should ensure that all threads which open a given file specify `POSIX_FADV_NOREUSE` to be sure that there is no unexpected interaction between threads using buffered I/O and threads using direct I/O to the same file.

In some cases, a user buffer must be properly aligned in order to be transferred directly to/from the device. The `{POSIX_REC_XFER_ALIGN}` *pathconf()* variable tells the application the proper alignment.

The preallocation goal is met by the space control function, *posix\_fallocate()*. The application can use *posix\_fallocate()* to guarantee no `[ENOSPC]` errors and to improve performance by prepaying any overhead required for block allocation.

Implementations may use information conveyed by a previous *posix\_fadvise()* call to influence the manner in which allocation is performed. For example, if an application did the following calls:

```
fd = open("file");
posix_fadvise(fd, offset, len, POSIX_FADV_SEQUENTIAL);
posix_fallocate(fd, len, size);
```

an implementation might allocate the file contiguously on disk.

Finally, the *pathconf()* variables `{POSIX_REC_MIN_XFER_SIZE}`, `{POSIX_REC_MAX_XFER_SIZE}`, and `{POSIX_REC_INCR_XFER_SIZE}` tell the application a range of transfer sizes that are recommended for best I/O performance.

Where bounded response time is required, the vendor can supply the appropriate settings of the advisories to achieve a guaranteed performance level.

The interfaces meet the goals while allowing applications using regular files to take advantage of performance optimizations. The interfaces tell the implementation expected application behavior which the implementation can use to optimize performance on a particular system with a particular dynamic load.

The *posix\_memalign()* function was added to allow for the allocation of specifically aligned buffers; for example, for {POSIX\_REC\_XFER\_ALIGN}.

The working group also considered the alternative of adding a function which would return an aligned pointer to memory within a user-supplied buffer. This was not considered to be the best method, because it potentially wastes large amounts of memory when buffers need to be aligned on large alignment boundaries.

## Message Passing

This section provides the rationale for the definition of the message passing interface in IEEE Std 1003.1-2001. This is presented in terms of the objectives, models, and requirements imposed upon this interface.

- Objectives

Many applications, including both realtime and database applications, require a means of passing arbitrary amounts of data between cooperating processes comprising the overall application on one or more processors. Many conventional interfaces for interprocess communication are insufficient for realtime applications in that efficient and deterministic data passing methods cannot be implemented. This has prompted the definition of message passing interfaces providing these facilities:

- Open a message queue.
- Send a message to a message queue.
- Receive a message from a queue, either synchronously or asynchronously.
- Alter message queue attributes for flow and resource control.

It is assumed that an application may consist of multiple cooperating processes and that these processes may wish to communicate and coordinate their activities. The message passing facility described in IEEE Std 1003.1-2001 allows processes to communicate through system-wide queues. These message queues are accessed through names that may be pathnames. A message queue can be opened for use by multiple sending and/or multiple receiving processes.

- Background on Embedded Applications

Interprocess communication utilizing message passing is a key facility for the construction of deterministic, high-performance realtime applications. The facility is present in all realtime systems and is the framework upon which the application is constructed. The performance of the facility is usually a direct indication of the performance of the resulting application.

Realtime applications, especially for embedded systems, are typically designed around the performance constraints imposed by the message passing mechanisms. Applications for embedded systems are typically very tightly constrained. Application writers expect to design and control the entire system. In order to minimize system costs, the writer will attempt to use all resources to their utmost and minimize the requirement to add additional memory or processors.

The embedded applications usually share address spaces and only a simple message passing mechanism is required. The application can readily access common data incurring only mutual-exclusion overheads. The models desired are the simplest possible with the application building higher-level facilities only when needed.

- Requirements

The following requirements determined the features of the message passing facilities defined in IEEE Std 1003.1-2001:

- Naming of Message Queues

The mechanism for gaining access to a message queue is a pathname evaluated in a context that is allowed to be a file system name space, or it can be independent of any file system. This is a specific attempt to allow implementations based on either method in order to address both embedded systems and to also allow implementation in larger systems.

The interface of *mq\_open()* is defined to allow but not require the access control and name conflicts resulting from utilizing a file system for name resolution. All required behavior is specified for the access control case. Yet a conforming implementation, such as an embedded system kernel, may define that there are no distinctions between users and may define that all processes have all access privileges.

- Embedded System Naming

Embedded systems need to be able to utilize independent name spaces for accessing the various system objects. They typically do not have a file system, precluding its utilization as a common name resolution mechanism. The modularity of an embedded system limits the connections between separate mechanisms that can be allowed.

Embedded systems typically do not have any access protection. Since the system does not support the mixing of applications from different areas, and usually does not even have the concept of an authorization entity, access control is not useful.

- Large System Naming

On systems with more functionality, the name resolution must support the ability to use the file system as the name resolution mechanism/object storage medium and to have control over access to the objects. Utilizing the pathname space can result in further errors when the names conflict with other objects.

- Fixed Size of Messages

The interfaces impose a fixed upper bound on the size of messages that can be sent to a specific message queue. The size is set on an individual queue basis and cannot be changed dynamically.

The purpose of the fixed size is to increase the ability of the system to optimize the implementation of *mq\_send()* and *mq\_receive()*. With fixed sizes of messages and fixed numbers of messages, specific message blocks can be pre-allocated. This eliminates a significant amount of checking for errors and boundary conditions. Additionally, an implementation can optimize data copying to maximize performance. Finally, with a restricted range of message sizes, an implementation is better able to provide deterministic operations.

- Prioritization of Messages

Message prioritization allows the application to determine the order in which messages are received. Prioritization of messages is a key facility that is provided by most realtime kernels and is heavily utilized by the applications. The major purpose of having priorities in message queues is to avoid priority inversions in the message system, where a high-priority message is delayed behind one or more lower-priority messages. This allows the applications to be designed so that they do not need to be interrupted in order to change

the flow of control when exceptional conditions occur. The prioritization does add additional overhead to the message operations in those cases it is actually used but a clever implementation can optimize for the FIFO case to make that more efficient.

#### — Asynchronous Notification

The interface supports the ability to have a task asynchronously notified of the availability of a message on the queue. The purpose of this facility is to allow the task to perform other functions and yet still be notified that a message has become available on the queue.

To understand the requirement for this function, it is useful to understand two models of application design: a single task performing multiple functions and multiple tasks performing a single function. Each of these models has advantages.

Asynchronous notification is required to build the model of a single task performing multiple operations. This model typically results from either the expectation that interruption is less expensive than utilizing a separate task or from the growth of the application to include additional functions.

### Semaphores

Semaphores are a high-performance process synchronization mechanism. Semaphores are named by null-terminated strings of characters.

A semaphore is created using the *sem\_init()* function or the *sem\_open()* function with the *O\_CREAT* flag set in *oflag*.

To use a semaphore, a process has to first initialize the semaphore or inherit an open descriptor for the semaphore via *fork()*.

A semaphore preserves its state when the last reference is closed. For example, if a semaphore has a value of 13 when the last reference is closed, it will have a value of 13 when it is next opened.

When a semaphore is created, an initial state for the semaphore has to be provided. This value is a non-negative integer. Negative values are not possible since they indicate the presence of blocked processes. The persistence of any of these objects across a system crash or a system reboot is undefined. Conforming applications must not depend on any sort of persistence across a system reboot or a system crash.

#### • Models and Requirements

A realtime system requires synchronization and communication between the processes comprising the overall application. An efficient and reliable synchronization mechanism has to be provided in a realtime system that will allow more than one schedulable process mutually-exclusive access to the same resource. This synchronization mechanism has to allow for the optimal implementation of synchronization or systems implementors will define other, more cost-effective methods.

At issue are the methods whereby multiple processes (tasks) can be designed and implemented to work together in order to perform a single function. This requires interprocess communication and synchronization. A semaphore mechanism is the lowest level of synchronization that can be provided by an operating system.

A semaphore is defined as an object that has an integral value and a set of blocked processes associated with it. If the value is positive or zero, then the set of blocked processes is empty; otherwise, the size of the set is equal to the absolute value of the semaphore value. The value of the semaphore can be incremented or decremented by any process with access to the

semaphore and must be done as an indivisible operation. When a semaphore value is less than or equal to zero, any process that attempts to lock it again will block or be informed that it is not possible to perform the operation.

A semaphore may be used to guard access to any resource accessible by more than one schedulable task in the system. It is a global entity and not associated with any particular process. As such, a method of obtaining access to the semaphore has to be provided by the operating system. A process that wants access to a critical resource (section) has to wait on the semaphore that guards that resource. When the semaphore is locked on behalf of a process, it knows that it can utilize the resource without interference by any other cooperating process in the system. When the process finishes its operation on the resource, leaving it in a well-defined state, it posts the semaphore, indicating that some other process may now obtain the resource associated with that semaphore.

In this section, mutexes and condition variables are specified as the synchronization mechanisms between threads.

These primitives are typically used for synchronizing threads that share memory in a single process. However, this section provides an option allowing the use of these synchronization interfaces and objects between processes that share memory, regardless of the method for sharing memory.

Much experience with semaphores shows that there are two distinct uses of synchronization: locking, which is typically of short duration; and waiting, which is typically of long or unbounded duration. These distinct usages map directly onto mutexes and condition variables, respectively.

Semaphores are provided in IEEE Std 1003.1-2001 primarily to provide a means of synchronization for processes; these processes may or may not share memory. Mutexes and condition variables are specified as synchronization mechanisms between threads; these threads always share (some) memory. Both are synchronization paradigms that have been in widespread use for a number of years. Each set of primitives is particularly well matched to certain problems.

With respect to binary semaphores, experience has shown that condition variables and mutexes are easier to use for many synchronization problems than binary semaphores. The primary reason for this is the explicit appearance of a Boolean predicate that specifies when the condition wait is satisfied. This Boolean predicate terminates a loop, including the call to *pthread\_cond\_wait()*. As a result, extra wakeups are benign since the predicate governs whether the thread will actually proceed past the condition wait. With stateful primitives, such as binary semaphores, the wakeup in itself typically means that the wait is satisfied. The burden of ensuring correctness for such waits is thus placed on *all* signalers of the semaphore rather than on an *explicitly coded* Boolean predicate located at the condition wait. Experience has shown that the latter creates a major improvement in safety and ease-of-use.

Counting semaphores are well matched to dealing with producer/consumer problems, including those that might exist between threads of different processes, or between a signal handler and a thread. In the former case, there may be little or no memory shared by the processes; in the latter case, one is not communicating between co-equal threads, but between a thread and an interrupt-like entity. It is for these reasons that IEEE Std 1003.1-2001 allows semaphores to be used by threads.

Mutexes and condition variables have been effectively used with and without priority inheritance, priority ceiling, and other attributes to synchronize threads that share memory. The efficiency of their implementation is comparable to or better than that of other synchronization primitives that are sometimes harder to use (for example, binary

semaphores). Furthermore, there is at least one known implementation of Ada tasking that uses these primitives. Mutexes and condition variables together constitute an appropriate, sufficient, and complete set of inter-thread synchronization primitives.

Efficient multi-threaded applications require high-performance synchronization primitives. Considerations of efficiency and generality require a small set of primitives upon which more sophisticated synchronization functions can be built.

- Standardization Issues

It is possible to implement very high-performance semaphores using test-and-set instructions on shared memory locations. The library routines that implement such a high-performance interface have to properly ensure that a *sem\_wait()* or *sem\_trywait()* operation that cannot be performed will issue a blocking semaphore system call or properly report the condition to the application. The same interface to the application program would be provided by a high-performance implementation.

#### B.2.8.1 Realtime Signals

##### Realtime Signals Extension

This portion of the rationale presents models, requirements, and standardization issues relevant to the Realtime Signals Extension. This extension provides the capability required to support reliable, deterministic, asynchronous notification of events. While a new mechanism, unencumbered by the historical usage and semantics of POSIX.1 signals, might allow for a more efficient implementation, the application requirements for event notification can be met with a small number of extensions to signals. Therefore, a minimal set of extensions to signals to support the application requirements is specified.

The realtime signal extensions specified in this section are used by other realtime functions requiring asynchronous notification:

- Models

The model supported is one of multiple cooperating processes, each of which handles multiple asynchronous external events. Events represent occurrences that are generated as the result of some activity in the system. Examples of occurrences that can constitute an event include:

- Completion of an asynchronous I/O request
- Expiration of a POSIX.1b timer
- Arrival of an interprocess message
- Generation of a user-defined event

Processing of these events may occur synchronously via polling for event notifications or asynchronously via a software interrupt mechanism. Existing practice for this model is well established for traditional proprietary realtime operating systems, realtime executives, and realtime extended POSIX-like systems.

A contrasting model is that of “cooperating sequential processes” where each process handles a single priority of events via polling. Each process blocks while waiting for events, and each process depends on the preemptive, priority-based process scheduling mechanism to arbitrate between events of different priority that need to be processed concurrently. Existing practice for this model is also well established for small realtime executives that typically execute in an unprotected physical address space, but it is just emerging in the

context of a fuller function operating system with multiple virtual address spaces.

It could be argued that the cooperating sequential process model, and the facilities supported by the POSIX Threads Extension obviate a software interrupt model. But, even with the cooperating sequential process model, the need has been recognized for a software interrupt model to handle exceptional conditions and process aborting, so the mechanism must be supported in any case. Furthermore, it is not the purview of IEEE Std 1003.1-2001 to attempt to convince realtime practitioners that their current application models based on software interrupts are “broken” and should be replaced by the cooperating sequential process model. Rather, it is the charter of IEEE Std 1003.1-2001 to provide standard extensions to mechanisms that support existing realtime practice.

- Requirements

This section discusses the following realtime application requirements for asynchronous event notification:

- Reliable delivery of asynchronous event notification

The events notification mechanism guarantees delivery of an event notification. Asynchronous operations (such as asynchronous I/O and timers) that complete significantly after they are invoked have to guarantee that delivery of the event notification can occur at the time of completion.

- Prioritized handling of asynchronous event notifications

The events notification mechanism supports the assigning of a user function as an event notification handler. Furthermore, the mechanism supports the preemption of an event handler function by a higher priority event notification and supports the selection of the highest priority pending event notification when multiple notifications (of different priority) are pending simultaneously.

The model here is based on hardware interrupts. Asynchronous event handling allows the application to ensure that time-critical events are immediately processed when delivered, without the indeterminism of being at a random location within a polling loop. Use of handler priority allows the specification of how handlers are interrupted by other higher priority handlers.

- Differentiation between multiple occurrences of event notifications of the same type

The events notification mechanism passes an application-defined value to the event handler function. This value can be used for a variety of purposes, such as enabling the application to identify which of several possible events of the same type (for example, timer expirations) has occurred.

- Polled reception of asynchronous event notifications

The events notification mechanism supports blocking and non-blocking polls for asynchronous event notification.

The polled mode of operation is often preferred over the interrupt mode by those practitioners accustomed to this model. Providing support for this model facilitates the porting of applications based on this model to POSIX.1b conforming systems.

- Deterministic response to asynchronous event notifications

The events notification mechanism does not preclude implementations that provide deterministic event dispatch latency and minimizes the number of system calls needed to use the event facilities during realtime processing.

- Rationale for Extension

POSIX.1 signals have many of the characteristics necessary to support the asynchronous handling of event notifications, and the Realtime Signals Extension addresses the following deficiencies in the POSIX.1 signal mechanism:

- Signals do not support reliable delivery of event notification. Subsequent occurrences of a pending signal are not guaranteed to be delivered.
- Signals do not support prioritized delivery of event notifications. The order of signal delivery when multiple unblocked signals are pending is undefined.
- Signals do not support the differentiation between multiple signals of the same type.

### B.2.8.2 Asynchronous I/O

Many applications need to interact with the I/O subsystem in an asynchronous manner. The asynchronous I/O mechanism provides the ability to overlap application processing and I/O operations initiated by the application. The asynchronous I/O mechanism allows a single process to perform I/O simultaneously to a single file multiple times or to multiple files multiple times.

#### Overview

Asynchronous I/O operations proceed in logical parallel with the processing done by the application after the asynchronous I/O has been initiated. Other than this difference, asynchronous I/O behaves similarly to normal I/O using *read()*, *write()*, *lseek()*, and *fsync()*. The effect of issuing an asynchronous I/O request is as if a separate thread of execution were to perform atomically the implied *lseek()* operation, if any, and then the requested I/O operation (either *read()*, *write()*, or *fsync()*). There is no seek implied with a call to *aio\_fsync()*. Concurrent asynchronous operations and synchronous operations applied to the same file update the file as if the I/O operations had proceeded serially.

When asynchronous I/O completes, a signal can be delivered to the application to indicate the completion of the I/O. This signal can be used to indicate that buffers and control blocks used for asynchronous I/O can be reused. Signal delivery is not required for an asynchronous operation and may be turned off on a per-operation basis by the application. Signals may also be synchronously polled using *aio\_suspend()*, *sigtimedwait()*, or *sigwaitinfo()*.

Normal I/O has a return value and an error status associated with it. Asynchronous I/O returns a value and an error status when the operation is first submitted, but that only relates to whether the operation was successfully queued up for servicing. The I/O operation itself also has a return status and an error value. To allow the application to retrieve the return status and the error value, functions are provided that, given the address of an asynchronous I/O control block, yield the return and error status associated with the operation. Until an asynchronous I/O operation is done, its error status is [EINPROGRESS]. Thus, an application can poll for completion of an asynchronous I/O operation by waiting for the error status to become equal to a value other than [EINPROGRESS]. The return status of an asynchronous I/O operation is undefined so long as the error status is equal to [EINPROGRESS].

Storage for asynchronous operation return and error status may be limited. Submission of asynchronous I/O operations may fail if this storage is exceeded. When an application retrieves the return status of a given asynchronous operation, therefore, any system-maintained storage used for this status and the error status may be reclaimed for use by other asynchronous operations.



Asynchronous I/O can be performed on file descriptors that have been enabled for POSIX.1b synchronized I/O. In this case, the I/O operation still occurs asynchronously, as defined herein; however, the asynchronous operation I/O in this case is not completed until the I/O has reached either the state of synchronized I/O data integrity completion or synchronized I/O file integrity completion, depending on the sort of synchronized I/O that is enabled on the file descriptor.

## Models

Three models illustrate the use of asynchronous I/O: a journalization model, a data acquisition model, and a model of the use of asynchronous I/O in supercomputing applications.

- Journalization Model

Many realtime applications perform low-priority journalizing functions. Journalizing requires that logging records be queued for output without blocking the initiating process.

- Data Acquisition Model

A data acquisition process may also serve as a model. The process has two or more channels delivering intermittent data that must be read within a certain time. The process issues one asynchronous read on each channel. When one of the channels needs data collection, the process reads the data and posts it through an asynchronous write to secondary memory for future processing.

- Supercomputing Model

The supercomputing community has used asynchronous I/O much like that specified in POSIX.1 for many years. This community requires the ability to perform multiple I/O operations to multiple devices with a minimal number of entries to “the system”; each entry to “the system” provokes a major delay in operations when compared to the normal progress made by the application. This existing practice motivated the use of combined *lseek()* and *read()* or *write()* calls, as well as the *lio\_listio()* call. Another common practice is to disable signal notification for I/O completion, and simply poll for I/O completion at some interval by which the I/O should be completed. Likewise, interfaces like *aio\_cancel()* have been in successful commercial use for many years. Note also that an underlying implementation of asynchronous I/O will require the ability, at least internally, to cancel outstanding asynchronous I/O, at least when the process exits. (Consider an asynchronous read from a terminal, when the process intends to exit immediately.)

## Requirements

Asynchronous input and output for realtime implementations have these requirements:

- The ability to queue multiple asynchronous read and write operations to a single open instance. Both sequential and random access should be supported.
- The ability to queue asynchronous read and write operations to multiple open instances.
- The ability to obtain completion status information by polling and/or asynchronous event notification.
- Asynchronous event notification on asynchronous I/O completion is optional.
- It has to be possible for the application to associate the event with the *aiocbp* for the operation that generated the event.
- The ability to cancel queued requests.
- The ability to wait upon asynchronous I/O completion in conjunction with other types of events.

- The ability to accept an *aio\_read()* and an *aio\_cancel()* for a device that accepts a *read()*, and the ability to accept an *aio\_write()* and an *aio\_cancel()* for a device that accepts a *write()*. This does not imply that the operation is asynchronous.

### Standardization Issues

The following issues are addressed by the standardization of asynchronous I/O:

- Rationale for New Interface

Non-blocking I/O does not satisfy the needs of either realtime or high-performance computing models; these models require that a process overlap program execution and I/O processing. Realtime applications will often make use of direct I/O to or from the address space of the process, or require synchronized (unbuffered) I/O; they also require the ability to overlap this I/O with other computation. In addition, asynchronous I/O allows an application to keep a device busy at all times, possibly achieving greater throughput. Supercomputing and database architectures will often have specialized hardware that can provide true asynchrony underlying the logical asynchrony provided by this interface. In addition, asynchronous I/O should be supported by all types of files and devices in the same manner.

- Effect of Buffering

If asynchronous I/O is performed on a file that is buffered prior to being actually written to the device, it is possible that asynchronous I/O will offer no performance advantage over normal I/O; the cycles *stolen* to perform the asynchronous I/O will be taken away from the running process and the I/O will occur at interrupt time. This potential lack of gain in performance in no way obviates the need for asynchronous I/O by realtime applications, which very often will use specialized hardware support, multiple processors, and/or unbuffered, synchronized I/O.

### B.2.8.3 Memory Management

All memory management and shared memory definitions are located in the `<sys/mman.h>` header. This is for alignment with historical practice.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/7 is applied, correcting the shading and margin markers in the introduction to Section 2.8.3.1.

### Memory Locking Functions

This portion of the rationale presents models, requirements, and standardization issues relevant to process memory locking.

- Models

Realtime systems that conform to IEEE Std 1003.1-2001 are expected (and desired) to be supported on systems with demand-paged virtual memory management, non-paged swapping memory management, and physical memory systems with no memory management hardware. The general case, however, is the demand-paged, virtual memory system with each POSIX process running in a virtual address space. Note that this includes architectures where each process resides in its own virtual address space and architectures where the address space of each process is only a portion of a larger global virtual address space.

The concept of memory locking is introduced to eliminate the indeterminacy introduced by paging and swapping, and to support an upper bound on the time required to access the memory mapped into the address space of a process. Ideally, this upper bound will be the

same as the time required for the processor to access “main memory”, including any address translation and cache miss overheads. But some implementations—primarily on mainframes—will not actually force locked pages to be loaded and held resident in main memory. Rather, they will handle locked pages so that accesses to these pages will meet the performance metrics for locked process memory in the implementation. Also, although it is not, for example, the intention that this interface, as specified, be used to lock process memory into “cache”, it is conceivable that an implementation could support a large static RAM memory and define this as “main memory” and use a large[r] dynamic RAM as “backing store”. These interfaces could then be interpreted as supporting the locking of process memory into the static RAM. Support for multiple levels of backing store would require extensions to these interfaces.

Implementations may also use memory locking to guarantee a fixed translation between virtual and physical addresses where such is beneficial to improving determinancy for direct-to/from-process input/output. IEEE Std 1003.1-2001 does not guarantee to the application that the virtual-to-physical address translations, if such exist, are fixed, because such behavior would not be implementable on all architectures on which implementations of IEEE Std 1003.1-2001 are expected. But IEEE Std 1003.1-2001 does mandate that an implementation define, for the benefit of potential users, whether or not locking guarantees fixed translations.

Memory locking is defined with respect to the address space of a process. Only the pages mapped into the address space of a process may be locked by the process, and when the pages are no longer mapped into the address space—for whatever reason—the locks established with respect to that address space are removed. Shared memory areas warrant special mention, as they may be mapped into more than one address space or mapped more than once into the address space of a process; locks may be established on pages within these areas with respect to several of these mappings. In such a case, the lock state of the underlying physical pages is the logical OR of the lock state with respect to each of the mappings. Only when all such locks have been removed are the shared pages considered unlocked.

In recognition of the page granularity of Memory Management Units (MMU), and in order to support locking of ranges of address space, memory locking is defined in terms of “page” granularity. That is, for the interfaces that support an address and size specification for the region to be locked, the address must be on a page boundary, and all pages mapped by the specified range are locked, if valid. This means that the length is implicitly rounded up to a multiple of the page size. The page size is implementation-defined and is available to applications as a compile-time symbolic constant or at runtime via *sysconf()*.

A “real memory” POSIX.1b implementation that has no MMU could elect not to support these interfaces, returning [ENOSYS]. But an application could easily interpret this as meaning that the implementation would unconditionally page or swap the application when such is not the case. It is the intention of IEEE Std 1003.1-2001 that such a system could define these interfaces as “NO-OPs”, returning success without actually performing any function except for mandated argument checking.

- Requirements

For realtime applications, memory locking is generally considered to be required as part of application initialization. This locking is performed after an application has been loaded (that is, *exec'd*) and the program remains locked for its entire lifetime. But to support applications that undergo major mode changes where, in one mode, locking is required, but in another it is not, the specified interfaces allow repeated locking and unlocking of memory within the lifetime of a process.

When a realtime application locks its address space, it should not be necessary for the application to then “touch” all of the pages in the address space to guarantee that they are resident or else suffer potential paging delays the first time the page is referenced. Thus, IEEE Std 1003.1-2001 requires that the pages locked by the specified interfaces be resident when the locking functions return successfully.

Many architectures support system-managed stacks that grow automatically when the current extent of the stack is exceeded. A realtime application has a requirement to be able to “preallocate” sufficient stack space and lock it down so that it will not suffer page faults to grow the stack during critical realtime operation. There was no consensus on a portable way to specify how much stack space is needed, so IEEE Std 1003.1-2001 supports no specific interface for preallocating stack space. But an application can portably lock down a specific amount of stack space by specifying `MCL_FUTURE` in a call to `mlockall()` and then calling a dummy function that declares an automatic array of the desired size.

Memory locking for realtime applications is also generally considered to be an “all or nothing” proposition. That is, the entire process, or none, is locked down. But, for applications that have well-defined sections that need to be locked and others that do not, IEEE Std 1003.1-2001 supports an optional set of interfaces to lock or unlock a range of process addresses. Reasons for locking down a specific range include:

- An asynchronous event handler function that must respond to external events in a deterministic manner such that page faults cannot be tolerated
- An input/output “buffer” area that is the target for direct-to-process I/O, and the overhead of implicit locking and unlocking for each I/O call cannot be tolerated

Finally, locking is generally viewed as an “application-wide” function. That is, the application is globally aware of which regions are locked and which are not over time. This is in contrast to a function that is used temporarily within a “third party” library routine whose function is unknown to the application, and therefore must have no “side effects”. The specified interfaces, therefore, do not support “lock stacking” or “lock nesting” within a process. But, for pages that are shared between processes or mapped more than once into a process address space, “lock stacking” is essentially mandated by the requirement that unlocking of pages that are mapped by more than one process or more than once by the same process does not affect locks established on the other mappings.

There was some support for “lock stacking” so that locking could be transparently used in functions or opaque modules. But the consensus was not to burden all implementations with lock stacking (and reference counting), and an implementation option was proposed. There were strong objections to the option because applications would have to support both options in order to remain portable. The consensus was to eliminate lock stacking altogether, primarily through overwhelming support for the System V “`m[un]lock[all]`” interface on which IEEE Std 1003.1-2001 is now based.

Locks are not inherited across `fork()`s because some implementations implement `fork()` by creating new address spaces for the child. In such an implementation, requiring locks to be inherited would lead to new situations in which a fork would fail due to the inability of the system to lock sufficient memory to lock both the parent and the child. The consensus was that there was no benefit to such inheritance. Note that this does not mean that locks are removed when, for instance, a thread is created in the same address space.

Similarly, locks are not inherited across `exec` because some implementations implement `exec` by unmapping all of the pages in the address space (which, by definition, removes the locks on these pages), and maps in pages of the `exec`’d image. In such an implementation, requiring locks to be inherited would lead to new situations in which `exec` would fail. Reporting this

failure would be very cumbersome to detect in time to report to the calling process, and no appropriate mechanism exists for informing the *exec'd* process of its status.

It was determined that, if the newly loaded application required locking, it was the responsibility of that application to establish the locks. This is also in keeping with the general view that it is the responsibility of the application to be aware of all locks that are established.

There was one request to allow (not mandate) locks to be inherited across *fork()*, and a request for a flag, *MCL\_INHERIT*, that would specify inheritance of memory locks across *execs*. Given the difficulties raised by this and the general lack of support for the feature in IEEE Std 1003.1-2001, it was not added. IEEE Std 1003.1-2001 does not preclude an implementation from providing this feature for administrative purposes, such as a “run” command that will lock down and execute a specified application. Additionally, the rationale for the objection equated *fork()* with creating a thread in the address space. IEEE Std 1003.1-2001 does not mandate releasing locks when creating additional threads in an existing process.

- Standardization Issues

One goal of IEEE Std 1003.1-2001 is to define a set of primitives that provide the necessary functionality for realtime applications, with consideration for the needs of other application domains where such were identified, which is based to the extent possible on existing industry practice.

The Memory Locking option is required by many realtime applications to tune performance. Such a facility is accomplished by placing constraints on the virtual memory system to limit paging of time of the process or of critical sections of the process. This facility should not be used by most non-realtime applications.

Optional features provided in IEEE Std 1003.1-2001 allow applications to lock selected address ranges with the caveat that the process is responsible for being aware of the page granularity of locking and the unnested nature of the locks.

## **Mapped Files Functions**

The Memory Mapped Files option provides a mechanism that allows a process to access files by directly incorporating file data into its address space. Once a file is “mapped” into a process address space, the data can be manipulated by instructions as memory. The use of mapped files can significantly reduce I/O data movement since file data does not have to be copied into process data buffers as in *read()* and *write()*. If more than one process maps a file, its contents are shared among them. This provides a low overhead mechanism by which processes can synchronize and communicate.

- Historical Perspective

Realtime applications have historically been implemented using a collection of cooperating processes or tasks. In early systems, these processes ran on bare hardware (that is, without an operating system) with no memory relocation or protection. The application paradigms that arose from this environment involve the sharing of data between the processes.

When realtime systems were implemented on top of vendor-supplied operating systems, the paradigm or performance benefits of direct access to data by multiple processes was still deemed necessary. As a result, operating systems that claim to support realtime applications must support the shared memory paradigm.

Additionally, a number of realtime systems provide the ability to map specific sections of the physical address space into the address space of a process. This ability is required if an

application is to obtain direct access to memory locations that have specific properties (for example, refresh buffers or display devices, dual ported memory locations, DMA target locations). The use of this ability is common enough to warrant some degree of standardization of its interface. This ability overlaps the general paradigm of shared memory in that, in both instances, common global objects are made addressable by individual processes or tasks.

Finally, a number of systems also provide the ability to map process addresses to files. This provides both a general means of sharing persistent objects, and using files in a manner that optimizes memory and swapping space usage.

Simple shared memory is clearly a special case of the more general file mapping capability. In addition, there is relatively widespread agreement and implementation of the file mapping interface. In these systems, many different types of objects can be mapped (for example, files, memory, devices, and so on) using the same mapping interfaces. This approach both minimizes interface proliferation and maximizes the generality of programs using the mapping interfaces.

- Memory Mapped Files Usage

A memory object can be concurrently mapped into the address space of one or more processes. The *mmap()* and *munmap()* functions allow a process to manipulate their address space by mapping portions of memory objects into it and removing them from it. When multiple processes map the same memory object, they can share access to the underlying data. Implementations may restrict the size and alignment of mappings to be on *page*-size boundaries. The page size, in bytes, is the value of the system-configurable variable {PAGESIZE}, typically accessed by calling *sysconf()* with a *name* argument of *\_SC\_PAGESIZE*. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

To map memory, a process first opens a memory object. The *ftruncate()* function can be used to contract or extend the size of the memory object even when the object is currently mapped. If the memory object is extended, the contents of the extended areas are zeros.

After opening a memory object, the application maps the object into its address space using the *mmap()* function call. Once a mapping has been established, it remains mapped until unmapped with *munmap()*, even if the memory object is closed. The *mprotect()* function can be used to change the memory protections initially established by *mmap()*.

A *close()* of the file descriptor, while invalidating the file descriptor itself, does not unmap any mappings established for the memory object. The address space, including all mapped regions, is inherited on *fork()*. The entire address space is unmapped on process termination or by successful calls to any of the *exec* family of functions.

The *msync()* function is used to force mapped file data to permanent storage.

- Effects on Other Functions

When the Memory Mapped Files option is supported, the operation of the *open()*, *creat()*, and *unlink()* functions are a natural result of using the file system name space to map the global names for memory objects.

The *ftruncate()* function can be used to set the length of a sharable memory object.

The meaning of *stat()* fields other than the size and protection information is undefined on implementations where memory objects are not implemented using regular files. When regular files are used, the times reflect when the implementation updated the file image of the data, not when a process updated the data in memory.

The operations of *fdopen()*, *write()*, *read()*, and *lseek()* were made unspecified for objects opened with *shm\_open()*, so that implementations that did not implement memory objects as regular files would not have to support the operation of these functions on shared memory objects.

The behavior of memory objects with respect to *close()*, *dup()*, *dup2()*, *open()*, *close()*, *fork()*, *\_exit()*, and the *exec* family of functions is the same as the behavior of the existing practice of the *mmap()* function.

A memory object can still be referenced after a close. That is, any mappings made to the file are still in effect, and reads and writes that are made to those mappings are still valid and are shared with other processes that have the same mapping. Likewise, the memory object can still be used if any references remain after its name(s) have been deleted. Any references that remain after a close must not appear to the application as file descriptors.

This is existing practice for *mmap()* and *close()*. In addition, there are already mappings present (text, data, stack) that do not have open file descriptors. The text mapping in particular is considered a reference to the file containing the text. The desire was to treat all mappings by the process uniformly. Also, many modern implementations use *mmap()* to implement shared libraries, and it would not be desirable to keep file descriptors for each of the many libraries an application can use. It was felt there were many other existing programs that used this behavior to free a file descriptor, and thus IEEE Std 1003.1-2001 could not forbid it and still claim to be using existing practice.

For implementations that implement memory objects using memory only, memory objects will retain the memory allocated to the file after the last close and will use that same memory on the next open. Note that closing the memory object is not the same as deleting the name, since the memory object is still defined in the memory object name space.

The locks of *fcntl()* do not block any read or write operation, including read or write access to shared memory or mapped files. In addition, implementations that only support shared memory objects should not be required to implement record locks. The reference to *fcntl()* is added to make this point explicitly. The other *fcntl()* commands are useful with shared memory objects.

The size of pages that mapping hardware may be able to support may be a configurable value, or it may change based on hardware implementations. The addition of the *\_SC\_PAGESIZE* parameter to the *sysconf()* function is provided for determining the mapping page size at runtime.

## Shared Memory Functions

Implementations may support the Shared Memory Objects option without supporting a general Memory Mapped Files option. Shared memory objects are named regions of storage that may be independent of the file system and can be mapped into the address space of one or more processes to allow them to share the associated memory.

- Requirements

Shared memory is used to share data among several processes, each potentially running at different priority levels, responding to different inputs, or performing separate tasks. Shared memory is not just simply providing common access to data, it is providing the fastest possible communication between the processes. With one memory write operation, a process can pass information to as many processes as have the memory region mapped.

As a result, shared memory provides a mechanism that can be used for all other interprocess communication facilities. It may also be used by an application for implementing more

sophisticated mechanisms than semaphores and message queues.

The need for a shared memory interface is obvious for virtual memory systems, where the operating system is directly preventing processes from accessing each other's data. However, in unprotected systems, such as those found in some embedded controllers, a shared memory interface is needed to provide a portable mechanism to allocate a region of memory to be shared and then to communicate the address of that region to other processes.

This, then, provides the minimum functionality that a shared memory interface must have in order to support realtime applications: to allocate and name an object to be mapped into memory for potential sharing (*open()* or *shm\_open()*), and to make the memory object available within the address space of a process (*mmap()*). To complete the interface, a mechanism to release the claim of a process on a shared memory object (*munmap()*) is also needed, as well as a mechanism for deleting the name of a sharable object that was previously created (*unlink()* or *shm\_unlink()*).

After a mapping has been established, an implementation should not have to provide services to maintain that mapping. All memory writes into that area will appear immediately in the memory mapping of that region by any other processes.

Thus, requirements include:

- Support creation of sharable memory objects and the mapping of these objects into the address space of a process.
- Sharable memory objects should be accessed by global names accessible from all processes.
- Support the mapping of specific sections of physical address space (such as a memory mapped device) into the address space of a process. This should not be done by the process specifying the actual address, but again by an implementation-defined global name (such as a special device name) dedicated to this purpose.
- Support the mapping of discrete portions of these memory objects.
- Support for minimum hardware configurations that contain no physical media on which to store shared memory contents permanently.
- The ability to preallocate the entire shared memory region so that minimum hardware configurations without virtual memory support can guarantee contiguous space.
- The maximizing of performance by not requiring functionality that would require implementation interaction above creating the shared memory area and returning the mapping.

Note that the above requirements do not preclude:

- The sharable memory object from being implemented using actual files on an actual file system.
- The global name that is accessible from all processes being restricted to a file system area that is dedicated to handling shared memory.
- An implementation not providing implementation-defined global names for the purpose of physical address mapping.

#### • Shared Memory Objects Usage

If the Shared Memory Objects option is supported, a shared memory object may be created, or opened if it already exists, with the *shm\_open()* function. If the shared memory object is created, it has a length of zero. The *ftruncate()* function can be used to set the size of the



shared memory object after creation. The *shm\_unlink()* function removes the name for a shared memory object created by *shm\_open()*.

- Shared Memory Overview

The shared memory facility defined by IEEE Std 1003.1-2001 usually results in memory locations being added to the address space of the process. The implementation returns the address of the new space to the application by means of a pointer. This works well in languages like C. However, in languages without pointer types it will not work. In the bindings for such a language, either a special COMMON section will need to be defined (which is unlikely), or the binding will have to allow existing structures to be mapped. The implementation will likely have to place restrictions on the size and alignment of such structures or will have to map a suitable region of the address space of the process into the memory object, and thus into other processes. These are issues for that particular language binding. For IEEE Std 1003.1-2001, however, the practice will not be forbidden, merely undefined.

Two potentially different name spaces are used for naming objects that may be mapped into process address spaces. When the Memory Mapped Files option is supported, files may be accessed via *open()*. When the Shared Memory Objects option is supported, sharable memory objects that might not be files may be accessed via the *shm\_open()* function. These options are not mutually-exclusive.

Some implementations supporting the Shared Memory Objects option may choose to implement the shared memory object name space as part of the file system name space. There are several reasons for this:

- It allows applications to prevent name conflicts by use of the directory structure.
- It uses an existing mechanism for accessing global objects and prevents the creation of a new mechanism for naming global objects.

In such implementations, memory objects can be implemented using regular files, if that is what the implementation chooses. The *shm\_open()* function can be implemented as an *open()* call in a fixed directory followed by a call to *fcntl()* to set FD\_CLOEXEC. The *shm\_unlink()* function can be implemented as an *unlink()* call.

On the other hand, it is also expected that small embedded systems that support the Shared Memory Objects option may wish to implement shared memory without having any file systems present. In this case, the implementations may choose to use a simple string valued name space for shared memory regions. The *shm\_open()* function permits either type of implementation.

Some implementations have hardware that supports protection of mapped data from certain classes of access and some do not. Systems that supply this functionality can support the Memory Protection option.

Some implementations restrict size, alignment, and protections to be on *page*-size boundaries. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size. Applications on implementations that do support larger pages must be cognizant of the page size since this is the alignment and protection boundary.

Simple embedded implementations may have a 1-byte page size and only support the Shared Memory Objects option. This provides simple shared memory between processes without requiring mapping hardware.

IEEE Std 1003.1-2001 specifically allows a memory object to remain referenced after a close because that is existing practice for the *mmap()* function.

## Typed Memory Functions

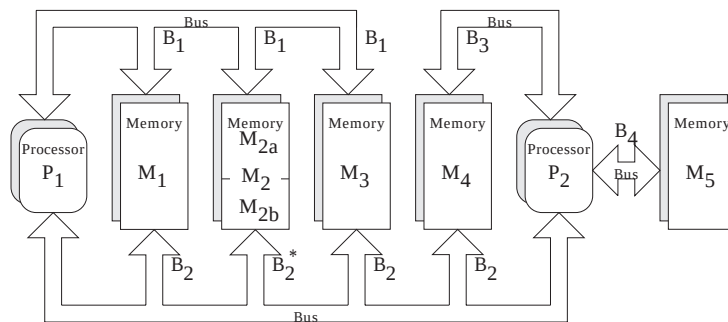
Implementations may support the Typed Memory Objects option without supporting either the Shared Memory option or the Memory Mapped Files option. Typed memory objects are pools of specialized storage, different from the main memory resource normally used by a processor to hold code and data, that can be mapped into the address space of one or more processes.

### • Model

Realtime systems conforming to one of the POSIX.13 realtime profiles are expected (and desired) to be supported on systems with more than one type or pool of memory (for example, SRAM, DRAM, ROM, EPROM, EEPROM), where each type or pool of memory may be accessible by one or more processors via one or more busses (ports). Memory mapped files, shared memory objects, and the language-specific storage allocation operators (*malloc()* for the ISO C standard, *new* for ISO Ada) fail to provide application program interfaces versatile enough to allow applications to control their utilization of such diverse memory resources. The typed memory interfaces *posix\_typed\_mem\_open()*, *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *mmap()*, and *munmap()* defined herein support the model of typed memory described below.

For purposes of this model, a system comprises several processors (for example,  $P_1$  and  $P_2$ ), several physical memory pools (for example,  $M_1$ ,  $M_2$ ,  $M_{2a}$ ,  $M_{2b}$ ,  $M_3$ ,  $M_4$ , and  $M_5$ ), and several busses or “ports” (for example,  $B_1$ ,  $B_2$ ,  $B_3$ , and  $B_4$ ) interconnecting the various processors and memory pools in some system-specific way. Notice that some memory pools may be contained in others (for example,  $M_{2a}$  and  $M_{2b}$  are contained in  $M_2$ ).

Figure B-1 shows an example of such a model. In a system like this, an application should be able to perform the following operations:



\* All addresses in pool  $M_2$  (comprising pools  $M_{2a}$  and  $M_{2b}$ ) accessible via port  $B_1$ .  
 Addresses in pool  $M_{2b}$  are also accessible via port  $B_2$ .  
 Addresses in pool  $M_{2a}$  are *not* accessible via port  $B_2$ .

**Figure B-1** Example of a System with Typed Memory

### — Typed Memory Allocation

An application should be able to allocate memory dynamically from the desired pool using the desired bus, and map it into a process' address space. For example, processor  $P_1$  can allocate some portion of memory pool  $M_1$  through port  $B_1$ , treating all unmapped subareas of  $M_1$  as a heap-storage resource from which memory may be allocated. This portion of memory is mapped into the process' address space, and subsequently deallocated when unmapped from all processes.

— Using the Same Storage Region from Different Busses

An application process with a mapped region of storage that is accessed from one bus should be able to map that same storage area at another address (subject to page size restrictions detailed in *mmap()*), to allow it to be accessed from another bus. For example, processor  $P_1$  may wish to access the same region of memory pool  $M_{2b}$  both through ports  $B_1$  and  $B_2$ .

— Sharing Typed Memory Regions

Several application processes running on the same or different processors may wish to share a particular region of a typed memory pool. Each process or processor may wish to access this region through different busses. For example, processor  $P_1$  may want to share a region of memory pool  $M_4$  with processor  $P_2$ , and they may be required to use busses  $B_2$  and  $B_3$ , respectively, to minimize bus contention. A problem arises here when a process allocates and maps a portion of fragmented memory and then wants to share this region of memory with another process, either in the same processor or different processors. The solution adopted is to allow the first process to find out the memory map (offsets and lengths) of all the different fragments of memory that were mapped into its address space, by repeatedly calling *posix\_mem\_offset()*. Then, this process can pass the offsets and lengths obtained to the second process, which can then map the same memory fragments into its address space.

— Contiguous Allocation

The problem of finding the memory map of the different fragments of the memory pool that were mapped into logically contiguous addresses of a given process can be solved by requesting contiguous allocation. For example, a process in  $P_1$  can allocate 10 Kbytes of physically contiguous memory from  $M_3-B_1$ , and obtain the offset (within pool  $M_3$ ) of this block of memory. Then, it can pass this offset (and the length) to a process in  $P_2$  using some interprocess communication mechanism. The second process can map the same block of memory by using the offset transferred and specifying  $M_3-B_2$ .

— Unallocated Mapping

Any subarea of a memory pool that is mapped to a process, either as the result of an allocation request or an explicit mapping, is normally unavailable for allocation. Special processes such as debuggers, however, may need to map large areas of a typed memory pool, yet leave those areas available for allocation.

Typed memory allocation and mapping has to coexist with storage allocation operators like *malloc()*, but systems are free to choose how to implement this coexistence. For example, it may be system configuration-dependent if all available system memory is made part of one of the typed memory pools or if some part will be restricted to conventional allocation operators. Equally system configuration-dependent may be the availability of operators like *malloc()* to allocate storage from certain typed memory pools. It is not excluded to configure a system such that a given named pool,  $P_1$ , is in turn split into non-overlapping named subpools. For example,  $M_1-B_1$ ,  $M_2-B_1$ , and  $M_3-B_1$  could also be accessed as one common pool  $M_{123}-B_1$ . A call to *malloc()* on  $P_1$  could work on such a larger pool while full optimization of memory usage by  $P_1$  would require typed memory allocation at the subpool level.

• Existing Practice

OS-9 provides for the naming (numbering) and prioritization of memory types by a system administrator. It then provides APIs to request memory allocation of typed (colored) memory by number, and to generate a bus address from a mapped memory address (translate). When requesting colored memory, the user can specify type 0 to signify allocation

from the first available type in priority order.

HP-RT presents interfaces to map different kinds of storage regions that are visible through a VME bus, although it does not provide allocation operations. It also provides functions to perform address translation between VME addresses and virtual addresses. It represents a VME-bus unique solution to the general problem.

The PSOS approach is similar (that is, based on a pre-established mapping of bus address ranges to specific memories) with a concept of segments and regions (regions dynamically allocated from a heap which is a special segment). Therefore, PSOS does not fully address the general allocation problem either. PSOS does not have a “process”-based model, but more of a “thread”-only-based model of multi-tasking. So mapping to a process address space is not an issue.

QNX uses the System V approach of opening specially named devices (shared memory segments) and using *mmap()* to then gain access from the process. They do not address allocation directly, but once typed shared memory can be mapped, an “allocation manager” process could be written to handle requests for allocation.

The System V approach also included allocation, implemented by opening yet other special “devices” which allocate, rather than appearing as a whole memory object.

The Orkid realtime kernel interface definition has operations to manage memory “regions” and “pools”, which are areas of memory that may reflect the differing physical nature of the memory. Operations to allocate memory from these regions and pools are also provided.

#### • Requirements

Existing practice in SVID-derived UNIX systems relies on functionality similar to *mmap()* and its related interfaces to achieve mapping and allocation of typed memory. However, the issue of sharing typed memory (allocated or mapped) and the complication of multiple ports are not addressed in any consistent way by existing UNIX system practice. Part of this functionality is existing practice in specialized realtime operating systems. In order to solidify the capabilities implied by the model above, the following requirements are imposed on the interface:

##### — Identification of Typed Memory Pools and Ports

All processes (running in all processors) in the system are able to identify a particular (system configured) typed memory pool accessed through a particular (system configured) port by a name. That name is a member of a name space common to all these processes, but need not be the same name space as that containing ordinary filenames. The association between memory pools/ports and corresponding names is typically established when the system is configured. The “open” operation for typed memory objects should be distinct from the *open()* function, for consistency with other similar services, but implementable on top of *open()*. This implies that the handle for a typed memory object will be a file descriptor.

##### — Allocation and Mapping of Typed Memory

Once a typed memory object has been identified by a process, it is possible to both map user-selected subareas of that object into process address space and to map system-selected (that is, dynamically allocated) subareas of that object, with user-specified length, into process address space. It is also possible to determine the maximum length of memory allocation that may be requested from a given typed memory object.

5134 — Sharing Typed Memory

5135 Two or more processes are able to share portions of typed memory, either user-selected or  
5136 dynamically allocated. This requirement applies also to dynamically allocated regions of  
5137 memory that are composed of several non-contiguous pieces.

5138 — Contiguous Allocation

5139 For dynamic allocation, it is the user's option whether the system is required to allocate a  
5140 contiguous subarea within the typed memory object, or whether it is permitted to allocate  
5141 discontinuous fragments which appear contiguous in the process mapping. Contiguous  
5142 allocation simplifies the process of sharing allocated typed memory, while discontinuous  
5143 allocation allows for potentially better recovery of deallocated typed memory.

5144 — Accessing Typed Memory Through Different Ports

5145 Once a subarea of a typed memory object has been mapped, it is possible to determine the  
5146 location and length corresponding to a user-selected portion of that object within the  
5147 memory pool. This location and length can then be used to remap that portion of memory  
5148 for access from another port. If the referenced portion of typed memory was allocated  
5149 discontinuously, the length thus determined may be shorter than anticipated, and the  
5150 user code must adapt to the value returned.

5151 — Deallocation

5152 When a previously mapped subarea of typed memory is no longer mapped by any  
5153 process in the system—as a result of a call or calls to *munmap()*—that subarea becomes  
5154 potentially reusable for dynamic allocation; actual reuse of the subarea is a function of the  
5155 dynamic typed memory allocation policy.

5156 — Unallocated Mapping

5157 It must be possible to map user-selected subareas of a typed memory object without  
5158 marking that subarea as unavailable for allocation. This option is not the default behavior,  
5159 and requires appropriate privilege.

5160 • Scenario

5161 The following scenario will serve to clarify the use of the typed memory interfaces.

5162 Process A running on  $P_1$  (see Figure B-1 (on page 122)) wants to allocate some memory from  
5163 memory pool  $M_2$ , and it wants to share this portion of memory with process B running on  $P_2$ .  
5164 Since  $P_2$  only has access to the lower part of  $M_2$ , both processes will use the memory pool  
5165 named  $M_{2b}$  which is the part of  $M_2$  that is accessible both from  $P_1$  and  $P_2$ . The operations that  
5166 both processes need to perform are shown below:

5167 — Allocating Typed Memory

5168 Process A calls *posix\_typed\_mem\_open()* with the name **/typed.m2b-b1** and a *tflag* of  
5169 `POSIX_TYPED_MEM_ALLOCATE` to get a file descriptor usable for allocating from pool  
5170  $M_{2b}$  accessed through port  $B_1$ . It then calls *mmap()* with this file descriptor requesting a  
5171 length of 4096 bytes. The system allocates two discontinuous blocks of sizes 1024 and  
5172 3072 bytes within  $M_{2b}$ . The *mmap()* function returns a pointer to a 4096-byte array in  
5173 process A's logical address space, mapping the allocated blocks contiguously. Process A  
5174 can then utilize the array, and store data in it.

5175 — Determining the Location of the Allocated Blocks

5176 Process A can determine the lengths and offsets (relative to  $M_{2b}$ ) of the two blocks  
5177 allocated, by using the following procedure: First, process A calls *posix\_mem\_offset()* with

the address of the first element of the array and length 4096. Upon return, the offset and length (1024 bytes) of the first block are returned. A second call to *posix\_mem\_offset()* is then made using the address of the first element of the array plus 1024 (the length of the first block), and a new length of 4096–1024. If there were more fragments allocated, this procedure could have been continued within a loop until the offsets and lengths of all the blocks were obtained. Notice that this relatively complex procedure can be avoided if contiguous allocation is requested (by opening the typed memory object with the *tflag* `POSIX_TYPED_MEM_ALLOCATE_CONTIG`).

— Sharing Data Across Processes

Process A passes the two offset values and lengths obtained from the *posix\_mem\_offset()* calls to process B running on  $P_2$ , via some form of interprocess communication. Process B can gain access to process A's data by calling *posix\_typed\_mem\_open()* with the name **/typed.m2b-b2** and a *tflag* of zero, then using two *mmap()* calls on the resulting file descriptor to map the two subareas of that typed memory object to its own address space.

- Rationale for no *mem\_alloc()* and *mem\_free()*

The standard developers had originally proposed a pair of new flags to *mmap()* which, when applied to a typed memory object descriptor, would cause *mmap()* to allocate dynamically from an unallocated and unmapped area of the typed memory object. Deallocation was similarly accomplished through the use of *munmap()*. This was rejected by the ballot group because it excessively complicated the (already rather complex) *mmap()* interface and introduced semantics useful only for typed memory, to a function which must also map shared memory and files. They felt that a memory allocator should be built on top of *mmap()* instead of being incorporated within the same interface, much as the ISO C standard libraries build *malloc()* on top of the virtual memory mapping functions *brk()* and *sbrk()*. This would eliminate the complicated semantics involved with unmapping only part of an allocated block of typed memory.

To attempt to achieve ballot group consensus, typed memory allocation and deallocation was first migrated from *mmap()* and *munmap()* to a pair of complementary functions modeled on the ISO C standard *malloc()* and *free()*. The *mem\_alloc()* function specified explicitly the typed memory object (typed memory pool/access port) from which allocation takes place, unlike *malloc()* where the memory pool and port are unspecified. The *mem\_free()* function handled deallocation. These new semantics still met all of the requirements detailed above without modifying the behavior of *mmap()* except to allow it to map specified areas of typed memory objects. An implementation would have been free to implement *mem\_alloc()* and *mem\_free()* over *mmap()*, through *mmap()*, or independently but cooperating with *mmap()*.

The ballot group was queried to see if this was an acceptable alternative, and while there was some agreement that it achieved the goal of removing the complicated semantics of allocation from the *mmap()* interface, several balloters realized that it just created two additional functions that behaved, in great part, like *mmap()*. These balloters proposed an alternative which has been implemented here in place of a separate *mem\_alloc()* and *mem\_free()*. This alternative is based on four specific suggestions:

1. The *posix\_typed\_mem\_open()* function should provide a flag which specifies “allocate on *mmap()*” (otherwise, *mmap()* just maps the underlying object). This allows things roughly similar to **/dev/zero** versus **/dev/swap**. Two such flags have been implemented, one of which forces contiguous allocation.
2. The *posix\_mem\_offset()* function is acceptable because it can be applied usefully to mapped objects in general. It should return the file descriptor of the underlying object.

3. The `mem_get_info()` function in an earlier draft should be renamed `posix_typed_mem_get_info()` because it is not generally applicable to memory objects. It should probably return the file descriptor's allocation attribute. The renaming of the function has been implemented, but having it return a piece of information which is readily known by an application without this function has been rejected. Its whole purpose is to query the typed memory object for attributes that are not user-specified, but determined by the implementation.

4. There should be no separate `mem_alloc()` or `mem_free()` functions. Instead, using `mmap()` on a typed memory object opened with an "allocate on `mmap()`" flag should be used to force allocation. These are precisely the semantics defined in the current draft.

- Rationale for no Typed Memory Access Management

The working group had originally defined an additional interface (and an additional kind of object: typed memory master) to establish and dissolve mappings to typed memory on behalf of devices or processors which were independent of the operating system and had no inherent capability to directly establish mappings on their own. This was to have provided functionality similar to device driver interfaces such as `physio()` and their underlying bus-specific interfaces (for example, `mballoc()`) which serve to set up and break down DMA pathways, and derive mapped addresses for use by hardware devices and processor cards.

The ballot group felt that this was beyond the scope of POSIX.1 and its amendments. Furthermore, the removal of interrupt handling interfaces from a preceding amendment (the IEEE Std 1003.1d-1999) during its balloting process renders these typed memory access management interfaces an incomplete solution to portable device management from a user process; it would be possible to initiate a device transfer to/from typed memory, but impossible to handle the transfer-complete interrupt in a portable way.

To achieve ballot group consensus, all references to typed memory access management capabilities were removed. The concept of portable interfaces from a device driver to both operating system and hardware is being addressed by the Uniform Driver Interface (UDI) industry forum, with formal standardization deferred until proof of concept and industry-wide acceptance and implementation.

#### B.2.8.4 Process Scheduling

IEEE PASC Interpretation 1003.1 #96 has been applied, adding the `pthread_setschedprio()` function. This was added since previously there was no way for a thread to lower its own priority without going to the tail of the threads list for its new priority. This capability is necessary to bound the duration of priority inversion encountered by a thread.

The following portion of the rationale presents models, requirements, and standardization issues relevant to process scheduling; see also Section B.2.9.4 (on page 167).

In an operating system supporting multiple concurrent processes, the system determines the order in which processes execute to meet implementation-defined goals. For time-sharing systems, the goal is to enhance system throughput and promote fairness; the application is provided with little or no control over this sequencing function. While this is acceptable and desirable behavior in a time-sharing system, it is inappropriate in a realtime system; realtime applications must specifically control the execution sequence of their concurrent processes in order to meet externally defined response requirements.

In IEEE Std 1003.1-2001, the control over process sequencing is provided using a concept of scheduling policies. These policies, described in detail in this section, define the behavior of the system whenever processor resources are to be allocated to competing processes. Only the behavior of the policy is defined; conforming implementations are free to use any mechanism

5272 desired to achieve the described behavior.

5273 • Models

5274 In an operating system supporting multiple concurrent processes, the system determines the  
5275 order in which processes execute and might force long-running processes to yield to other  
5276 processes at certain intervals. Typically, the scheduling code is executed whenever an event  
5277 occurs that might alter the process to be executed next.

5278 The simplest scheduling strategy is a “first-in, first-out” (FIFO) dispatcher. Whenever a  
5279 process becomes runnable, it is placed on the end of a ready list. The process at the front of  
5280 the ready list is executed until it exits or becomes blocked, at which point it is removed from  
5281 the list. This scheduling technique is also known as “run-to-completion” or “run-to-block”.

5282 A natural extension to this scheduling technique is the assignment of a “non-migrating  
5283 priority” to each process. This policy differs from strict FIFO scheduling in only one respect:  
5284 whenever a process becomes runnable, it is placed at the end of the list of processes runnable  
5285 at that priority level. When selecting a process to run, the system always selects the first  
5286 process from the highest priority queue with a runnable process. Thus, when a process  
5287 becomes unblocked, it will preempt a running process of lower priority without otherwise  
5288 altering the ready list. Further, if a process elects to alter its priority, it is removed from the  
5289 ready list and reinserted, using its new priority, according to the policy above.

5290 While the above policy might be considered unfriendly in a time-sharing environment in  
5291 which multiple users require more balanced resource allocation, it could be ideal in a  
5292 realtime environment for several reasons. The most important of these is that it is  
5293 deterministic: the highest-priority process is always run and, among processes of equal  
5294 priority, the process that has been runnable for the longest time is executed first. Because of  
5295 this determinism, cooperating processes can implement more complex scheduling simply by  
5296 altering their priority. For instance, if processes at a single priority were to reschedule  
5297 themselves at fixed time intervals, a time-slice policy would result.

5298 In a dedicated operating system in which all processes are well-behaved realtime  
5299 applications, non-migrating priority scheduling is sufficient. However, many existing  
5300 implementations provide for more complex scheduling policies.

5301 IEEE Std 1003.1-2001 specifies a linear scheduling model. In this model, every process in the  
5302 system has a priority. The system scheduler always dispatches a process that has the highest  
5303 (generally the most time-critical) priority among all runnable processes in the system. As  
5304 long as there is only one such process, the dispatching policy is trivial. When multiple  
5305 processes of equal priority are eligible to run, they are ordered according to a strict run-to-  
5306 completion (FIFO) policy.

5307 The priority is represented as a positive integer and is inherited from the parent process. For  
5308 processes running under a fixed priority scheduling policy, the priority is never altered  
5309 except by an explicit function call.

5310 It was determined arbitrarily that larger integers correspond to “higher priorities”.

5311 Certain implementations might impose restrictions on the priority ranges to which processes  
5312 can be assigned. There also can be restrictions on the set of policies to which processes can be  
5313 set.

5314 • Requirements

5315 Realtime processes require that scheduling be fast and deterministic, and that it guarantees  
5316 to preempt lower priority processes.



Thus, given the linear scheduling model, realtime processes require that they be run at a priority that is higher than other processes. Within this framework, realtime processes are free to yield execution resources to each other in a completely portable and implementation-defined manner.

As there is a generally perceived requirement for processes at the same priority level to share processor resources more equitably, provisions are made by providing a scheduling policy (that is, SCHED\_RR) intended to provide a timeslice-like facility.

**Note:** The following topics assume that low numeric priority implies low scheduling criticality and *vice versa*.

- Rationale for New Interface

Realtime applications need to be able to determine when processes will run in relation to each other. It must be possible to guarantee that a critical process will run whenever it is runnable; that is, whenever it wants to for as long as it needs. SCHED\_FIFO satisfies this requirement. Additionally, SCHED\_RR was defined to meet a realtime requirement for a well-defined time-sharing policy for processes at the same priority.

It would be possible to use the BSD *setpriority()* and *getpriority()* functions by redefining the meaning of the “nice” parameter according to the scheduling policy currently in use by the process. The System V *nice()* interface was felt to be undesirable for realtime because it specifies an adjustment to the “nice” value, rather than setting it to an explicit value. Realtime applications will usually want to set priority to an explicit value. Also, System V *nice()* does not allow for changing the priority of another process.

With the POSIX.1b interfaces, the traditional “nice” value does not affect the SCHED\_FIFO or SCHED\_RR scheduling policies. If a “nice” value is supported, it is implementation-defined whether it affects the SCHED\_OTHER policy.

An important aspect of IEEE Std 1003.1-2001 is the explicit description of the queuing and preemption rules. It is critical, to achieve deterministic scheduling, that such rules be stated clearly in IEEE Std 1003.1-2001.

IEEE Std 1003.1-2001 does not address the interaction between priority and swapping. The issues involved with swapping and virtual memory paging are extremely implementation-defined and would be nearly impossible to standardize at this point. The proposed scheduling paradigm, however, fully describes the scheduling behavior of runnable processes, of which one criterion is that the working set be resident in memory. Assuming the existence of a portable interface for locking portions of a process in memory, paging behavior need not affect the scheduling of realtime processes.

IEEE Std 1003.1-2001 also does not address the priorities of “system” processes. In general, these processes should always execute in low-priority ranges to avoid conflict with other realtime processes. Implementations should document the priority ranges in which system processes run.

The default scheduling policy is not defined. The effect of I/O interrupts and other system processing activities is not defined. The temporary lending of priority from one process to another (such as for the purposes of affecting freeing resources) by the system is not addressed. Preemption of resources is not addressed. Restrictions on the ability of a process to affect other processes beyond a certain level (influence levels) is not addressed.

The rationale used to justify the simple time-quantum scheduler is that it is common practice to depend upon this type of scheduling to ensure “fair” distribution of processor resources among portions of the application that must interoperate in a serial fashion. Note that IEEE Std 1003.1-2001 is silent with respect to the setting of this time quantum, or whether it is

a system-wide value or a per-process value, although it appears that the prevailing realtime practice is for it to be a system-wide value.

In a system with  $N$  processes at a given priority, all processor-bound, in which the time quantum is equal for all processes at a specific priority level, the following assumptions are made of such a scheduling policy:

1. A time quantum  $Q$  exists and the current process will own control of the processor for at least a duration of  $Q$  and will have the processor for a duration of  $Q$ .
2. The  $N$ th process at that priority will control a processor within a duration of  $(N-1) \times Q$ .

These assumptions are necessary to provide equal access to the processor and bounded response from the application.

The assumptions hold for the described scheduling policy only if no system overhead, such as interrupt servicing, is present. If the interrupt servicing load is non-zero, then one of the two assumptions becomes fallacious, based upon how  $Q$  is measured by the system.

If  $Q$  is measured by clock time, then the assumption that the process obtains a duration  $Q$  processor time is false if interrupt overhead exists. Indeed, a scenario can be constructed with  $N$  processes in which a single process undergoes complete processor starvation if a peripheral device, such as an analog-to-digital converter, generates significant interrupt activity periodically with a period of  $N \times Q$ .

If  $Q$  is measured as actual processor time, then the assumption that the  $N$ th process runs in within the duration  $(N-1) \times Q$  is false.

It should be noted that SCHED\_FIFO suffers from interrupt-based delay as well. However, for SCHED\_FIFO, the implied response of the system is “as soon as possible”, so that the interrupt load for this case is a vendor selection and not a compliance issue.

With this in mind, it is necessary either to complete the definition by including bounds on the interrupt load, or to modify the assumptions that can be made about the scheduling policy.

Since the motivation of inclusion of the policy is common usage, and since current applications do not enjoy the luxury of bounded interrupt load, item (2) above is sufficient to express existing application needs and is less restrictive in the standard definition. No difference in interface is necessary.

In an implementation in which the time quantum is equal for all processes at a specific priority, our assumptions can then be restated as:

- A time quantum  $Q$  exists, and a processor-bound process will be rescheduled after a duration of, at most,  $Q$ . Time quantum  $Q$  may be defined in either wall clock time or execution time.
- In general, the  $N$ th process of a priority level should wait no longer than  $(N-1) \times Q$  time to execute, assuming no processes exist at higher priority levels.
- No process should wait indefinitely.

For implementations supporting per-process time quanta, these assumptions can be readily extended.

## Sporadic Server Scheduling Policy

The sporadic server is a mechanism defined for scheduling aperiodic activities in time-critical realtime systems. This mechanism reserves a certain bounded amount of execution capacity for processing aperiodic events at a high priority level. Any aperiodic events that cannot be processed within the bounded amount of execution capacity are executed in the background at a low priority level. Thus, a certain amount of execution capacity can be guaranteed to be available for processing periodic tasks, even under burst conditions in the arrival of aperiodic processing requests (that is, a large number of requests in a short time interval). The sporadic server also simplifies the schedulability analysis of the realtime system, because it allows aperiodic processes or threads to be treated as if they were periodic. The sporadic server was first described by Sprunt, et al.

The key concept of the sporadic server is to provide and limit a certain amount of computation capacity for processing aperiodic events at their assigned normal priority, during a time interval called the “replenishment period”. Once the entity controlled by the sporadic server mechanism is initialized with its period and execution-time budget attributes, it preserves its execution capacity until an aperiodic request arrives. The request will be serviced (if there are no higher priority activities pending) as long as there is execution capacity left. If the request is completed, the actual execution time used to service it is subtracted from the capacity, and a replenishment of this amount of execution time is scheduled to happen one replenishment period after the arrival of the aperiodic request. If the request is not completed, because there is no execution capacity left, then the aperiodic process or thread is assigned a lower background priority. For each portion of consumed execution capacity the execution time used is replenished after one replenishment period. At the time of replenishment, if the sporadic server was executing at a background priority level, its priority is elevated to the normal level. Other similar replenishment policies have been defined, but the one presented here represents a compromise between efficiency and implementation complexity.

The interface that appears in this section defines a new scheduling policy for threads and processes that behaves according to the rules of the sporadic server mechanism. Scheduling attributes are defined and functions are provided to allow the user to set and get the parameters that control the scheduling behavior of this mechanism, namely the normal and low priority, the replenishment period, the maximum number of pending replenishment operations, and the initial execution-time budget.

- Scheduling Aperiodic Activities

Virtually all realtime applications are required to process aperiodic activities. In many cases, there are tight timing constraints that the response to the aperiodic events must meet. Usual timing requirements imposed on the response to these events are:

- The effects of an aperiodic activity on the response time of lower priority activities must be controllable and predictable.
- The system must provide the fastest possible response time to aperiodic events.
- It must be possible to take advantage of all the available processing bandwidth not needed by time-critical activities to enhance average-case response times to aperiodic events.

Traditional methods for scheduling aperiodic activities are background processing, polling tasks, and direct event execution:

- Background processing consists of assigning a very low priority to the processing of aperiodic events. It utilizes all the available bandwidth in the system that has not been consumed by higher priority threads. However, it is very difficult, or impossible, to meet

5450 requirements on average-case response time, because the aperiodic entity has to wait for  
5451 the execution of all other entities which have higher priority.

5452 — Polling consists of creating a periodic process or thread for servicing aperiodic requests.  
5453 At regular intervals, the polling entity is started and its services accumulated pending  
5454 aperiodic requests. If no aperiodic requests are pending, the polling entity suspends itself  
5455 until its next period. Polling allows the aperiodic requests to be processed at a higher  
5456 priority level. However, worst and average-case response times of polling entities are a  
5457 direct function of the polling period, and there is execution overhead for each polling  
5458 period, even if no event has arrived. If the deadline of the aperiodic activity is short  
5459 compared to the inter-arrival time, the polling frequency must be increased to guarantee  
5460 meeting the deadline. For this case, the increase in frequency can dramatically reduce the  
5461 efficiency of the system and, therefore, its capacity to meet all deadlines. Yet, polling  
5462 represents a good way to handle a large class of practical problems because it preserves  
5463 system predictability, and because the amortized overhead drops as load increases.

5464 — Direct event execution consists of executing the aperiodic events at a high fixed-priority  
5465 level. Typically, the aperiodic event is processed by an interrupt service routine as soon as  
5466 it arrives. This technique provides predictable response times for aperiodic events, but  
5467 makes the response times of all lower priority activities completely unpredictable under  
5468 burst arrival conditions. Therefore, if the density of aperiodic event arrivals is  
5469 unbounded, it may be a dangerous technique for time-critical systems. Yet, for those cases  
5470 in which the physics of the system imposes a bound on the event arrival rate, it is  
5471 probably the most efficient technique.

5472 — The sporadic server scheduling algorithm combines the predictability of the polling  
5473 approach with the short response times of the direct event execution. Thus, it allows  
5474 systems to meet an important class of application requirements that cannot be met by  
5475 using the traditional approaches. Multiple sporadic servers with different attributes can  
5476 be applied to the scheduling of multiple classes of aperiodic events, each with different  
5477 kinds of timing requirements, such as individual deadlines, average response times, and  
5478 so on. It also has many other interesting applications for realtime, such as scheduling  
5479 producer/consumer tasks in time-critical systems, limiting the effects of faults on the  
5480 estimation of task execution-time requirements, and so on.

#### 5481 • Existing Practice

5482 The sporadic server has been used in different kinds of applications, including military  
5483 avionics, robot control systems, industrial automation systems, and so on. There are  
5484 examples of many systems that cannot be successfully scheduled using the classic  
5485 approaches, such as direct event execution, or polling, and are schedulable using a sporadic  
5486 server scheduler. The sporadic server algorithm itself can successfully schedule all systems  
5487 scheduled with direct event execution or polling.

5488 The sporadic server scheduling policy has been implemented as a commercial product in the  
5489 run-time system of the Verdix Ada compiler. There are also many applications that have  
5490 used a much less efficient application-level sporadic server. These realtime applications  
5491 would benefit from a sporadic server scheduler implemented at the scheduler level.

#### 5492 • Library-Level versus Kernel-Level Implementation

5493 The sporadic server interface described in this section requires the sporadic server policy to  
5494 be implemented at the same level as the scheduler. This means that the process sporadic  
5495 server must be implemented at the kernel level and the thread sporadic server policy  
5496 implemented at the same level as the thread scheduler; that is, kernel or library level.

In an earlier interface for the sporadic server, this mechanism was implementable at a different level than the scheduler. This feature allowed the implementor to choose between an efficient scheduler-level implementation, or a simpler user or library-level implementation. However, the working group considered that this interface made the use of sporadic servers more complex, and that library-level implementations would lack some of the important functionality of the sporadic server, namely the limitation of the actual execution time of aperiodic activities. The working group also felt that the interface described in this chapter does not preclude library-level implementations of threads intended to provide efficient low-overhead scheduling for those threads that are not scheduled under the sporadic server policy.

- Range of Scheduling Priorities

Each of the scheduling policies supported in IEEE Std 1003.1-2001 has an associated range of priorities. The priority ranges for each policy might or might not overlap with the priority ranges of other policies. For time-critical realtime applications it is usual for periodic and aperiodic activities to be scheduled together in the same processor. Periodic activities will usually be scheduled using the SCHED\_FIFO scheduling policy, while aperiodic activities may be scheduled using SCHED\_SPORADIC. Since the application developer will require complete control over the relative priorities of these activities in order to meet his timing requirements, it would be desirable for the priority ranges of SCHED\_FIFO and SCHED\_SPORADIC to overlap completely. Therefore, although IEEE Std 1003.1-2001 does not require any particular relationship between the different priority ranges, it is recommended that these two ranges should coincide.

- Dynamically Setting the Sporadic Server Policy

Several members of the working group requested that implementations should not be required to support dynamically setting the sporadic server scheduling policy for a thread. The reason is that this policy may have a high overhead for library-level implementations of threads, and if threads are allowed to dynamically set this policy, this overhead can be experienced even if the thread does not use that policy. By disallowing the dynamic setting of the sporadic server scheduling policy, these implementations can accomplish efficient scheduling for threads using other policies. If a strictly conforming application needs to use the sporadic server policy, and is therefore willing to pay the overhead, it must set this policy at the time of thread creation.

- Limitation of the Number of Pending Replenishments

The number of simultaneously pending replenishment operations must be limited for each sporadic server for two reasons: an unlimited number of replenishment operations would need an unlimited number of system resources to store all the pending replenishment operations; on the other hand, in some implementations each replenishment operation will represent a source of priority inversion (just for the duration of the replenishment operation) and thus, the maximum amount of replenishments must be bounded to guarantee bounded response times. The way in which the number of replenishments is bounded is by lowering the priority of the sporadic server to *sched\_ss\_low\_priority* when the number of pending replenishments has reached its limit. In this way, no new replenishments are scheduled until the number of pending replenishments decreases.

In the sporadic server scheduling policy defined in IEEE Std 1003.1-2001, the application can specify the maximum number of pending replenishment operations for a single sporadic server, by setting the value of the *sched\_ss\_max\_repl* scheduling parameter. This value must be between one and {SS\_REPL\_MAX}, which is a maximum limit imposed by the implementation. The limit {SS\_REPL\_MAX} must be greater than or equal to {\_POSIX\_SS\_REPL\_MAX}, which is defined to be four in IEEE Std 1003.1-2001. The minimum

5546 limit of four was chosen so that an application can at least guarantee that four different  
 5547 aperiodic events can be processed during each interval of length equal to the replenishment  
 5548 period.

#### 5549 B.2.8.5 Clocks and Timers

##### 5550 • Clocks

5551 IEEE Std 1003.1-2001 and the ISO C standard both define functions for obtaining system time.  
 5552 Implicit behind these functions is a mechanism for measuring passage of time. This  
 5553 specification makes this mechanism explicit and calls it a clock. The `CLOCK_REALTIME`  
 5554 clock required by IEEE Std 1003.1-2001 is a higher resolution version of the clock that  
 5555 maintains POSIX.1 system time. This is a “system-wide” clock, in that it is visible to all  
 5556 processes and, were it possible for multiple processes to all read the clock at the same time,  
 5557 they would see the same value.

5558 An extensible interface was defined, with the ability for implementations to define additional  
 5559 clocks. This was done because of the observation that many realtime platforms support  
 5560 multiple clocks, and it was desired to fit this model within the standard interface. But  
 5561 implementation-defined clocks need not represent actual hardware devices, nor are they  
 5562 necessarily system-wide.

##### 5563 • Timers

5564 Two timer types are required for a system to support realtime applications:

##### 5565 1. One-shot

5566 A one-shot timer is a timer that is armed with an initial expiration time, either relative  
 5567 to the current time or at an absolute time (based on some timing base, such as time in  
 5568 seconds and nanoseconds since the Epoch). The timer expires once and then is  
 5569 disarmed. With the specified facilities, this is accomplished by setting the *it\_value*  
 5570 member of the *value* argument to the desired expiration time and the *it\_interval* member  
 5571 to zero.

##### 5572 2. Periodic

5573 A periodic timer is a timer that is armed with an initial expiration time, again either  
 5574 relative or absolute, and a repetition interval. When the initial expiration occurs, the  
 5575 timer is reloaded with the repetition interval and continues counting. With the  
 5576 specified facilities, this is accomplished by setting the *it\_value* member of the *value*  
 5577 argument to the desired initial expiration time and the *it\_interval* member to the desired  
 5578 repetition interval.

5579 For both of these types of timers, the time of the initial timer expiration can be specified in  
 5580 two ways:

##### 5581 1. Relative (to the current time)

##### 5582 2. Absolute

##### 5583 • Examples of Using Realtime Timers

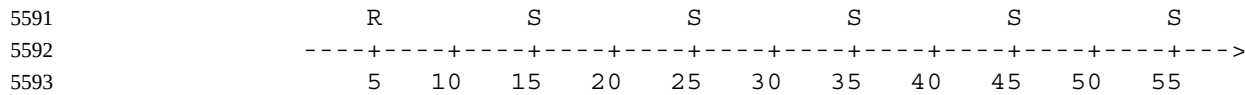
5584 In the diagrams below, *S* indicates a program schedule, *R* shows a schedule method request,  
 5585 and *E* suggests an internal operating system event.

##### 5586 — Periodic Timer: Data Logging

5587 During an experiment, it might be necessary to log realtime data periodically to an  
 5588 internal buffer or to a mass storage device. With a periodic scheduling method, a logging

5589 module can be started automatically at fixed time intervals to log the data.

5590 Program schedule is requested every 10 seconds.



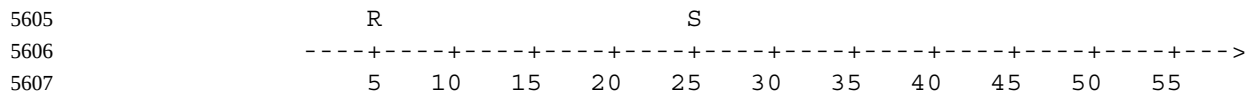
5594 [Time (in Seconds)]

5595 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
5596 process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via  
5597 a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an initial  
5598 expiration value and a repetition interval of 10 seconds.

5599 — One-shot Timer (Relative Time): Device Initialization

5600 In an emission test environment, large sample bags are used to capture the exhaust from  
5601 a vehicle. The exhaust is purged from these bags before each and every test. With a one-  
5602 shot timer, a module could initiate the purge function and then suspend itself for a  
5603 predetermined period of time while the sample bags are prepared.

5604 Program schedule requested 20 seconds after call is issued.



5608 [Time (in Seconds)]

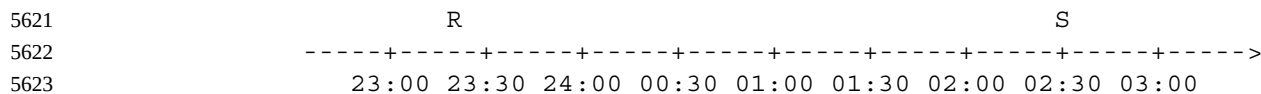
5609 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
5610 process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via  
5611 a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an initial  
5612 expiration value of 20 seconds and a repetition interval of zero.

5613 Note that if the program wishes merely to suspend itself for the specified interval, it  
5614 could more easily use `nanosleep()`.

5615 — One-shot Timer (Absolute Time): Data Transmission

5616 The results from an experiment are often moved to a different system within a network  
5617 for postprocessing or archiving. With an absolute one-shot timer, a module that moves  
5618 data from a test-cell computer to a host computer can be automatically scheduled on a  
5619 daily basis.

5620 Program schedule requested for 2:30 a.m.



5624 [Time of Day]

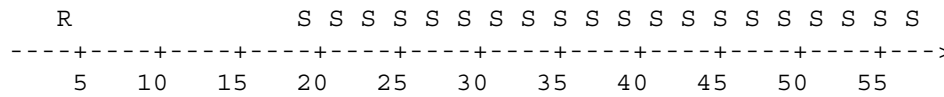
5625 To achieve this type of scheduling using the specified facilities, a per-process timer would  
5626 be allocated based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via  
5627 a call to `timer_settime()` with the `TIMER_ABSTIME` flag set, and an initial expiration value  
5628 equal to 2:30 a.m. of the next day.

5629 — Periodic Timer (Relative Time): Signal Stabilization

5630 Some measurement devices, such as emission analyzers, do not respond instantaneously  
5631 to an introduced sample. With a periodic timer with a relative initial expiration time, a

module that introduces a sample and records the average response could suspend itself for a predetermined period of time while the signal is stabilized and then sample at a fixed rate.

Program schedule requested 15 seconds after call is issued and every 2 seconds thereafter.



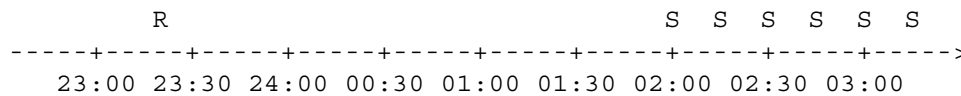
[Time (in Seconds)]

To achieve this type of scheduling using the specified facilities, one would allocate a per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via a call to `timer_settime()` with `TIMER_ABSTIME` flag reset, and with an initial expiration value of 15 seconds and a repetition interval of 2 seconds.

#### — Periodic Timer (Absolute Time): Work Shift-related Processing

Resource utilization data is useful when time to perform experiments is being scheduled at a facility. With a periodic timer with an absolute initial expiration time, a module can be scheduled at the beginning of a work shift to gather resource utilization data throughout the shift. This data can be used to allocate resources effectively to minimize bottlenecks and delays and maximize facility throughput.

Program schedule requested for 2:00 a.m. and every 15 minutes thereafter.



[Time of Day]

To achieve this type of scheduling using the specified facilities, one would allocate a per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via a call to `timer_settime()` with `TIMER_ABSTIME` flag set, and with an initial expiration value equal to 2:00 a.m. and a repetition interval equal to 15 minutes.

#### • Relationship of Timers to Clocks

The relationship between clocks and timers armed with an absolute time is straightforward: a timer expiration signal is requested when the associated clock reaches or exceeds the specified time. The relationship between clocks and timers armed with a relative time (an interval) is less obvious, but not unintuitive. In this case, a timer expiration signal is requested when the specified interval, *as measured by the associated clock*, has passed. For the required `CLOCK_REALTIME` clock, this allows timer expiration signals to be requested at specified “wall clock” times (absolute), or when a specified interval of “realtime” has passed (relative). For an implementation-defined clock—say, a process virtual time clock—timer expirations could be requested when the process has used a specified total amount of virtual time (absolute), or when it has used a specified *additional* amount of virtual time (relative).

The interfaces also allow flexibility in the implementation of the functions. For example, an implementation could convert all absolute times to intervals by subtracting the clock value at the time of the call from the requested expiration time and “counting down” at the supported resolution. Or it could convert all relative times to absolute expiration time by adding in the clock value at the time of the call and comparing the clock value to the expiration time at the supported resolution. Or it might even choose to maintain absolute times as absolute and compare them to the clock value at the supported resolution for absolute timers, and maintain relative times as intervals and count them down at the



5678 resolution supported for relative timers. The choice will be driven by efficiency  
 5679 considerations and the underlying hardware or software clock implementation.

5680 • Data Definitions for Clocks and Timers

5681 IEEE Std 1003.1-2001 uses a time representation capable of supporting nanosecond resolution  
 5682 timers for the following reasons:

5683 — To enable IEEE Std 1003.1-2001 to represent those computer systems already using  
 5684 nanosecond or submicrosecond resolution clocks.

5685 — To accommodate those per-process timers that might need nanoseconds to specify an  
 5686 absolute value of system-wide clocks, even though the resolution of the per-process timer  
 5687 may only be milliseconds, or *vice versa*.

5688 — Because the number of nanoseconds in a second can be represented in 32 bits.

5689 Time values are represented in the **timespec** structure. The *tv\_sec* member is of type **time\_t**  
 5690 so that this member is compatible with time values used by POSIX.1 functions and the ISO C  
 5691 standard. The *tv\_nsec* member is a **signed long** in order to simplify and clarify code that  
 5692 decrements or finds differences of time values. Note that because 1 billion (number of  
 5693 nanoseconds per second) is less than half of the value representable by a signed 32-bit value,  
 5694 it is always possible to add two valid fractional seconds represented as integral nanoseconds  
 5695 without overflowing the signed 32-bit value.

5696 A maximum allowable resolution for the CLOCK\_REALTIME clock of 20 ms (1/50 seconds)  
 5697 was chosen to allow line frequency clocks in European countries to be conforming. 60 Hz  
 5698 clocks in the U.S. will also be conforming, as will finer granularity clocks, although a Strictly  
 5699 Conforming Application cannot assume a granularity of less than 20 ms (1/50 seconds).

5700 The minimum allowable maximum time allowed for the CLOCK\_REALTIME clock and the  
 5701 function *nanosleep()*, and timers created with *clock\_id*=CLOCK\_REALTIME, is determined by  
 5702 the fact that the *tv\_sec* member is of type **time\_t**.

5703 IEEE Std 1003.1-2001 specifies that timer expirations must not be delivered early, and  
 5704 *nanosleep()* must not return early due to quantization error. IEEE Std 1003.1-2001 discusses  
 5705 the various implementations of *alarm()* in the rationale and states that implementations that  
 5706 do not allow alarm signals to occur early are the most appropriate, but refrained from  
 5707 mandating this behavior. Because of the importance of predictability to realtime applications,  
 5708 IEEE Std 1003.1-2001 takes a stronger stance.

5709 The developers of IEEE Std 1003.1-2001 considered using a time representation that differs  
 5710 from POSIX.1b in the second 32 bit of the 64-bit value. Whereas POSIX.1b defines this field  
 5711 as a fractional second in nanoseconds, the other methodology defines this as a binary fraction  
 5712 of one second, with the radix point assumed before the most significant bit.

5713 POSIX.1b is a software, source-level standard and most of the benefits of the alternate  
 5714 representation are enjoyed by hardware implementations of clocks and algorithms. It was  
 5715 felt that mandating this format for POSIX.1b clocks and timers would unnecessarily burden  
 5716 the application writer with writing, possibly non-portable, multiple precision arithmetic  
 5717 packages to perform conversion between binary fractions and integral units such as  
 5718 nanoseconds, milliseconds, and so on.

**Rationale for the Monotonic Clock**

For those applications that use time services to achieve realtime behavior, changing the value of the clock on which these services rely may cause erroneous timing behavior. For these applications, it is necessary to have a monotonic clock which cannot run backwards, and which has a maximum clock jump that is required to be documented by the implementation. Additionally, it is desirable (but not required by IEEE Std 1003.1-2001) that the monotonic clock increases its value uniformly. This clock should not be affected by changes to the system time; for example, to synchronize the clock with an external source or to account for leap seconds. Such changes would cause errors in the measurement of time intervals for those time services that use the absolute value of the clock.

One could argue that by defining the behavior of time services when the value of a clock is changed, deterministic realtime behavior can be achieved. For example, one could specify that relative time services should be unaffected by changes in the value of a clock. However, there are time services that are based upon an absolute time, but that are essentially intended as relative time services. For example, *pthread\_cond\_timedwait()* uses an absolute time to allow it to wake up after the required interval despite spurious wakeups. Although sometimes the *pthread\_cond\_timedwait()* timeouts are absolute in nature, there are many occasions in which they are relative, and their absolute value is determined from the current time plus a relative time interval. In this latter case, if the clock changes while the thread is waiting, the wait interval will not be the expected length. If a *pthread\_cond\_timedwait()* function were created that would take a relative time, it would not solve the problem because to retain the intended “deadline” a thread would need to compensate for latency due to the spurious wakeup, and preemption between wakeup and the next wait.

The solution is to create a new monotonic clock, whose value does not change except for the regular ticking of the clock, and use this clock for implementing the various relative timeouts that appear in the different POSIX interfaces, as well as allow *pthread\_cond\_timedwait()* to choose this new clock for its timeout. A new *clock\_nanosleep()* function is created to allow an application to take advantage of this newly defined clock. Notice that the monotonic clock may be implemented using the same hardware clock as the system clock.

Relative timeouts for *sigtimedwait()* and *aio\_suspend()* have been redefined to use the monotonic clock, if present. The *alarm()* function has not been redefined, because the same effect but with better resolution can be achieved by creating a timer (for which the appropriate clock may be chosen).

The *pthread\_cond\_timedwait()* function has been treated in a different way, compared to other functions with absolute timeouts, because it is used to wait for an event, and thus it may have a deadline, while the other timeouts are generally used as an error recovery mechanism, and for them the use of the monotonic clock is not so important. Since the desired timeout for the *pthread\_cond\_timedwait()* function may either be a relative interval or an absolute time of day deadline, a new initialization attribute has been created for condition variables to specify the clock that is used for measuring the timeout in a call to *pthread\_cond\_timedwait()*. In this way, if a relative timeout is desired, the monotonic clock will be used; if an absolute deadline is required instead, the *CLOCK\_REALTIME* or another appropriate clock may be used. This capability has not been added to other functions with absolute timeouts because for those functions the expected use of the timeout is mostly to prevent errors, and not so often to meet precise deadlines. As a consequence, the complexity of adding this capability is not justified by its perceived application usage.

The *nanosleep()* function has not been modified with the introduction of the monotonic clock. Instead, a new *clock\_nanosleep()* function has been created, in which the desired clock may be specified in the function call.

- History of Resolution Issues

Due to the shift from relative to absolute timeouts in IEEE Std 1003.1d-1999, the amendments to the `sem_timedwait()`, `pthread_mutex_timedlock()`, `mq_timedreceive()`, and `mq_timedsend()` functions of that standard have been removed. Those amendments specified that `CLOCK_MONOTONIC` would be used for the (relative) timeouts if the Monotonic Clock option was supported.

Having these functions continue to be tied solely to `CLOCK_MONOTONIC` would not work. Since the absolute value of a time value obtained from `CLOCK_MONOTONIC` is unspecified, under the absolute timeouts interface, applications would behave differently depending on whether the Monotonic Clock option was supported or not (because the absolute value of the clock would have different meanings in either case).

Two options were considered:

1. Leave the current behavior unchanged, which specifies the `CLOCK_REALTIME` clock for these (absolute) timeouts, to allow portability of applications between implementations supporting or not the Monotonic Clock option.
2. Modify these functions in the way that `pthread_cond_timedwait()` was modified to allow a choice of clock, so that an application could use `CLOCK_REALTIME` when it is trying to achieve an absolute timeout and `CLOCK_MONOTONIC` when it is trying to achieve a relative timeout.

It was decided that the features of `CLOCK_MONOTONIC` are not as critical to these functions as they are to `pthread_cond_timedwait()`. The `pthread_cond_timedwait()` function is given a relative timeout; the timeout may represent a deadline for an event. When these functions are given relative timeouts, the timeouts are typically for error recovery purposes and need not be so precise.

Therefore, it was decided that these functions should be tied to `CLOCK_REALTIME` and not complicated by being given a choice of clock.

## Execution Time Monitoring

- Introduction

The main goals of the execution time monitoring facilities defined in this chapter are to measure the execution time of processes and threads and to allow an application to establish CPU time limits for these entities.

The analysis phase of time-critical realtime systems often relies on the measurement of execution times of individual threads or processes to determine whether the timing requirements will be met. Also, performance analysis techniques for soft deadline realtime systems rely heavily on the determination of these execution times. The execution time monitoring functions provide application developers with the ability to measure these execution times online and open the possibility of dynamic execution-time analysis and system reconfiguration, if required.

The second goal of allowing an application to establish execution time limits for individual processes or threads and detecting when they overrun allows program robustness to be increased by enabling online checking of the execution times.

If errors are detected—possibly because of erroneous program constructs, the existence of errors in the analysis phase, or a burst of event arrivals—online detection and recovery is possible in a portable way. This feature can be extremely important for many time-critical applications. Other applications require trapping CPU-time errors as a normal way to exit an

algorithm; for instance, some realtime artificial intelligence applications trigger a number of independent inference processes of varying accuracy and speed, limit how long they can run, and pick the best answer available when time runs out. In many periodic systems, overrun processes are simply restarted in the next resource period, after necessary end-of-period actions have been taken. This allows algorithms that are inherently data-dependent to be made predictable.

The interface that appears in this chapter defines a new type of clock, the CPU-time clock, which measures execution time. Each process or thread can invoke the clock and timer functions defined in POSIX.1 to use them. Functions are also provided to access the CPU-time clock of other processes or threads to enable remote monitoring of these clocks. Monitoring of threads of other processes is not supported, since these threads are not visible from outside of their own process with the interfaces defined in POSIX.1.

- Execution Time Monitoring Interface

The clock and timer interface defined in POSIX.1 historically only defined one clock, which measures wall-clock time. The requirements for measuring execution time of processes and threads, and setting limits to their execution time by detecting when they overrun, can be accomplished with that interface if a new kind of clock is defined. These new clocks measure execution time, and one is associated with each process and with each thread. The clock functions currently defined in POSIX.1 can be used to read and set these CPU-time clocks, and timers can be created using these clocks as their timing base. These timers can then be used to send a signal when some specified execution time has been exceeded. The CPU-time clocks of each process or thread can be accessed by using the symbols `CLOCK_PROCESS_CPUTIME_ID` or `CLOCK_THREAD_CPUTIME_ID`.

The clock and timer interface defined in POSIX.1 and extended with the new kind of CPU-time clock would only allow processes or threads to access their own CPU-time clocks. However, many realtime systems require the possibility of monitoring the execution time of processes or threads from independent monitoring entities. In order to allow applications to construct independent monitoring entities that do not require cooperation from or modification of the monitored entities, two functions have been added: `clock_getcpuclockid()`, for accessing CPU-time clocks of other processes, and `pthread_getcpuclockid()`, for accessing CPU-time clocks of other threads. These functions return the clock identifier associated with the process or thread specified in the call. These clock IDs can then be used in the rest of the clock function calls.

The clocks accessed through these functions could also be used as a timing base for the creation of timers, thereby allowing independent monitoring entities to limit the CPU time consumed by other entities. However, this possibility would imply additional complexity and overhead because of the need to maintain a timer queue for each process or thread, to store the different expiration times associated with timers created by different processes or threads. The working group decided this additional overhead was not justified by application requirements. Therefore, creation of timers attached to the CPU-time clocks of other processes or threads has been specified as implementation-defined.

- Overhead Considerations

The measurement of execution time may introduce additional overhead in the thread scheduling, because of the need to keep track of the time consumed by each of these entities. In library-level implementations of threads, the efficiency of scheduling could be somehow compromised because of the need to make a kernel call, at each context switch, to read the process CPU-time clock. Consequently, a thread creation attribute called *cpu-clock-requirement* was defined, to allow threads to disconnect their respective CPU-time clocks. However, the Ballot Group considered that this attribute itself introduced some overhead,

and that in current implementations it was not worth the effort. Therefore, the attribute was deleted, and thus thread CPU-time clocks are required for all threads if the Thread CPU-Time Clocks option is supported.

- Accuracy of CPU-Time Clocks

The mechanism used to measure the execution time of processes and threads is specified in IEEE Std 1003.1-2001 as implementation-defined. The reason for this is that both the underlying hardware and the implementation architecture have a very strong influence on the accuracy achievable for measuring CPU time. For some implementations, the specification of strict accuracy requirements would represent very large overheads, or even the impossibility of being implemented.

Since the mechanism for measuring execution time is implementation-defined, realtime applications will be able to take advantage of accurate implementations using a portable interface. Of course, strictly conforming applications cannot rely on any particular degree of accuracy, in the same way as they cannot rely on a very accurate measurement of wall clock time. There will always exist applications whose accuracy or efficiency requirements on the implementation are more rigid than the values defined in IEEE Std 1003.1-2001 or any other standard.

In any case, there is a minimum set of characteristics that realtime applications would expect from most implementations. One such characteristic is that the sum of all the execution times of all the threads in a process equals the process execution time, when no CPU-time clocks are disabled. This need not always be the case because implementations may differ in how they account for time during context switches. Another characteristic is that the sum of the execution times of all processes in a system equals the number of processors, multiplied by the elapsed time, assuming that no processor is idle during that elapsed time. However, in some implementations it might not be possible to relate CPU time to elapsed time. For example, in a heterogeneous multi-processor system in which each processor runs at a different speed, an implementation may choose to define each “second” of CPU time to be a certain number of “cycles” that a CPU has executed.

- Existing Practice

Measuring and limiting the execution time of each concurrent activity are common features of most industrial implementations of realtime systems. Almost all critical realtime systems are currently built upon a cyclic executive. With this approach, a regular timer interrupt kicks off the next sequence of computations. It also checks that the current sequence has completed. If it has not, then some error recovery action can be undertaken (or at least an overrun is avoided). Current software engineering principles and the increasing complexity of software are driving application developers to implement these systems on multi-threaded or multi-process operating systems. Therefore, if a POSIX operating system is to be used for this type of application, then it must offer the same level of protection.

Execution time clocks are also common in most UNIX implementations, although these clocks usually have requirements different from those of realtime applications. The POSIX.1 *times()* function supports the measurement of the execution time of the calling process, and its terminated child processes. This execution time is measured in clock ticks and is supplied as two different values with the user and system execution times, respectively. BSD supports the function *getrusage()*, which allows the calling process to get information about the resources used by itself and/or all of its terminated child processes. The resource usage includes user and system CPU time. Some UNIX systems have options to specify high resolution (up to one microsecond) CPU-time clocks using the *times()* or the *getrusage()* functions.

The *times()* and *getrusage()* interfaces do not meet important realtime requirements, such as the possibility of monitoring execution time from a different process or thread, or the possibility of detecting an execution time overrun. The latter requirement is supported in some UNIX implementations that are able to send a signal when the execution time of a process has exceeded some specified value. For example, BSD defines the functions *getitimer()* and *setitimer()*, which can operate either on a realtime clock (wall-clock), or on virtual-time or profile-time clocks which measure CPU time in two different ways. These functions do not support access to the execution time of other processes.

IBM's MVS operating system supports per-process and per-thread execution time clocks. It also supports limiting the execution time of a given process.

Given all this existing practice, the working group considered that the POSIX.1 clocks and timers interface was appropriate to meet most of the requirements that realtime applications have for execution time clocks. Functions were added to get the CPU time clock IDs, and to allow/disallow the thread CPU-time clocks (in order to preserve the efficiency of some implementations of threads).

- Clock Constants

The definition of the manifest constants `CLOCK_PROCESS_CPUTIME_ID` and `CLOCK_THREAD_CPUTIME_ID` allows processes or threads, respectively, to access their own execution-time clocks. However, given a process or thread, access to its own execution-time clock is also possible if the clock ID of this clock is obtained through a call to *clock\_getcpuclockid()* or *pthread\_getcpuclockid()*. Therefore, these constants are not necessary and could be deleted to make the interface simpler. Their existence saves one system call in the first access to the CPU-time clock of each process or thread. The working group considered this issue and decided to leave the constants in IEEE Std 1003.1-2001 because they are closer to the POSIX.1b use of clock identifiers.

- Library Implementations of Threads

In library implementations of threads, kernel entities and library threads can coexist. In this case, if the CPU-time clocks are supported, most of the clock and timer functions will need to have two implementations: one in the thread library, and one in the system calls library. The main difference between these two implementations is that the thread library implementation will have to deal with clocks and timers that reside in the thread space, while the kernel implementation will operate on timers and clocks that reside in kernel space. In the library implementation, if the clock ID refers to a clock that resides in the kernel, a kernel call will have to be made. The correct version of the function can be chosen by specifying the appropriate order for the libraries during the link process.

- History of Resolution Issues: Deletion of the *enable* Attribute

In early proposals, consideration was given to inclusion of an attribute called *enable* for CPU-time clocks. This would allow implementations to avoid the overhead of measuring execution time for those processes or threads for which this measurement was not required. However, this is unnecessary since processes are already required to measure execution time by the POSIX.1 *times()* function. Consequently, the *enable* attribute is not present.

**Rationale Relating to Timeouts**

- Requirements for Timeouts

Realtime systems which must operate reliably over extended periods without human intervention are characteristic in embedded applications such as avionics, machine control, and space exploration, as well as more mundane applications such as cable TV, security systems, and plant automation. A multi-tasking paradigm, in which many independent and/or cooperating software functions relinquish the processor(s) while waiting for a specific stimulus, resource, condition, or operation completion, is very useful in producing well engineered programs for such systems. For such systems to be robust and fault-tolerant, expected occurrences that are unduly delayed or that never occur must be detected so that appropriate recovery actions may be taken. This is difficult if there is no way for a task to regain control of a processor once it has relinquished control (blocked) awaiting an occurrence which, perhaps because of corrupted code, hardware malfunction, or latent software bugs, will not happen when expected. Therefore, the common practice in realtime operating systems is to provide a capability to time out such blocking services. Although there are several methods to achieve this already defined by POSIX, none are as reliable or efficient as initiating a timeout simultaneously with initiating a blocking service. This is especially critical in hard-realtime embedded systems because the processors typically have little time reserve, and allowed fault recovery times are measured in milliseconds rather than seconds.

The working group largely agreed that such timeouts were necessary and ought to become part of IEEE Std 1003.1-2001, particularly vendors of realtime operating systems whose customers had already expressed a strong need for timeouts. There was some resistance to inclusion of timeouts in IEEE Std 1003.1-2001 because the desired effect, fault tolerance, could, in theory, be achieved using existing facilities and alternative software designs, but there was no compelling evidence that realtime system designers would embrace such designs at the sacrifice of performance and/or simplicity.

- Which Services should be Timed Out?

Originally, the working group considered the prospect of providing timeouts on all blocking services, including those currently existing in POSIX.1, POSIX.1b, and POSIX.1c, and future interfaces to be defined by other working groups, as sort of a general policy. This was rather quickly rejected because of the scope of such a change, and the fact that many of those services would not normally be used in a realtime context. More traditional timesharing solutions to timeout would suffice for most of the POSIX.1 interfaces, while others had asynchronous alternatives which, while more complex to utilize, would be adequate for some realtime and all non-realtime applications.

The list of potential candidates for timeouts was narrowed to the following for further consideration:

- POSIX.1b

- `sem_wait()`

- `mq_receive()`

- `mq_send()`

- `lio_listio()`

- `aio_suspend()`

- `sigwait()` (timeout already implemented by `sigtimedwait()`)

5996 — POSIX.1c  
 5997 — *pthread\_mutex\_lock()*  
 5998 — *pthread\_join()*  
 5999 — *pthread\_cond\_wait()* (timeout already implemented by *pthread\_cond\_timedwait()*)  
 6000 — POSIX.1  
 6001 — *read()*  
 6002 — *write()*

6003 After further review by the working group, the *lio\_listio()*, *read()*, and *write()* functions (all  
 6004 forms of blocking synchronous I/O) were eliminated from the list because of the following:

6005 — Asynchronous alternatives exist  
 6006 — Timeouts can be implemented, albeit non-portably, in device drivers  
 6007 — A strong desire not to introduce modifications to POSIX.1 interfaces

6008 The working group ultimately rejected *pthread\_join()* since both that interface and a timed  
 6009 variant of that interface are non-minimal and may be implemented as a function. See below  
 6010 for a library implementation of *pthread\_join()*.

6011 Thus, there was a consensus among the working group members to add timeouts to 4 of the  
 6012 remaining 5 functions (the timeout for *aio\_suspend()* was ultimately added directly to  
 6013 POSIX.1b, while the others were added by POSIX.1d). However, *pthread\_mutex\_lock()*  
 6014 remained contentious.

6015 Many feel that *pthread\_mutex\_lock()* falls into the same class as the other functions; that is, it  
 6016 is desirable to time out a mutex lock because a mutex may fail to be unlocked due to errant or  
 6017 corrupted code in a critical section (looping or branching outside of the unlock code), and  
 6018 therefore is equally in need of a reliable, simple, and efficient timeout. In fact, since mutexes  
 6019 are intended to guard small critical sections, most *pthread\_mutex\_lock()* calls would be  
 6020 expected to obtain the lock without blocking nor utilizing any kernel service, even in  
 6021 implementations of threads with global contention scope; the timeout alternative need only  
 6022 be considered after it is determined that the thread must block.

6023 Those opposed to timing out mutexes feel that the very simplicity of the mutex is  
 6024 compromised by adding a timeout semantic, and that to do so is senseless. They claim that if  
 6025 a timed mutex is really deemed useful by a particular application, then it can be constructed  
 6026 from the facilities already in POSIX.1b and POSIX.1c. The following two C-language library  
 6027 implementations of mutex locking with timeout represent the solutions offered (in both  
 6028 implementations, the timeout parameter is specified as absolute time, not relative time as in  
 6029 the proposed POSIX.1c interfaces).



- Spinlock Implementation

```

6030
6031 #include <pthread.h>
6032 #include <time.h>
6033 #include <errno.h>
6034
6035 int pthread_mutex_timedlock(pthread_mutex_t *mutex,
6036 const struct timespec *timeout)
6037 {
6038 struct timespec timenow;
6039
6040 while (pthread_mutex_trylock(mutex) == EBUSY)
6041 {
6042 clock_gettime(CLOCK_REALTIME, &timenow);
6043 if (timespec_cmp(&timenow, timeout) >= 0)
6044 {
6045 return ETIMEDOUT;
6046 }
6047 pthread_yield();
6048 }
6049 return 0;
6050 }

```

The Spinlock implementation is generally unsuitable for any application using priority-based thread scheduling policies such as SCHED\_FIFO or SCHED\_RR, since the mutex could currently be held by a thread of lower priority within the same allocation domain, but since the waiting thread never blocks, only threads of equal or higher priority will ever run, and the mutex cannot be unlocked. Setting priority inheritance or priority ceiling protocol on the mutex does not solve this problem, since the priority of a mutex owning thread is only boosted if higher priority threads are blocked waiting for the mutex; clearly not the case for this spinlock.

- Condition Wait Implementation

```

6057
6058 #include <pthread.h>
6059 #include <time.h>
6060 #include <errno.h>
6061
6062 struct timed_mutex
6063 {
6064 int locked;
6065 pthread_mutex_t mutex;
6066 pthread_cond_t cond;
6067 };
6068
6069 typedef struct timed_mutex timed_mutex_t;
6070
6071 int timed_mutex_lock(timed_mutex_t *tm,
6072 const struct timespec *timeout)
6073 {
6074 int timedout=FALSE;
6075 int error_status;
6076
6077 pthread_mutex_lock(&tm->mutex);
6078
6079 while (tm->locked && !timedout)
6080 {
6081 if ((error_status=pthread_cond_timedwait(&tm->cond,

```

```

6077 &tm->mutex,
6078 timeout)) != 0)
6079 {
6080 if (error_status==ETIMEDOUT) timedout = TRUE;
6081 }
6082 }
6083
6084 if(timedout)
6085 {
6086 pthread_mutex_unlock(&tm->mutex);
6087 return ETIMEDOUT;
6088 }
6089 else
6090 {
6091 tm->locked = TRUE;
6092 pthread_mutex_unlock(&tm->mutex);
6093 return 0;
6094 }
6095
6096 void timed_mutex_unlock(timed_mutex_t *tm)
6097 {
6098 pthread_mutex_lock(&tm->mutex); / for case assignment not atomic /
6099 tm->locked = FALSE;
6100 pthread_mutex_unlock(&tm->mutex);
6101 pthread_cond_signal(&tm->cond);
6102 }

```

The Condition Wait implementation effectively substitutes the *pthread\_cond\_timedwait()* function (which is currently timed out) for the desired *pthread\_mutex\_timedlock()*. Since waits on condition variables currently do not include protocols which avoid priority inversion, this method is generally unsuitable for realtime applications because it does not provide the same priority inversion protection as the untimed *pthread\_mutex\_lock()*. Also, for any given implementations of the current mutex and condition variable primitives, this library implementation has a performance cost at least 2.5 times that of the untimed *pthread\_mutex\_lock()* even in the case where the timed mutex is readily locked without blocking (the interfaces required for this case are shown in bold). Even in uniprocessors or where assignment is atomic, at least an additional *pthread\_cond\_signal()* is required. *pthread\_mutex\_timedlock()* could be implemented at effectively no performance penalty in this case because the timeout parameters need only be considered after it is determined that the mutex cannot be locked immediately.

Thus it has not yet been shown that the full semantics of mutex locking with timeout can be efficiently and reliably achieved using existing interfaces. Even if the existence of an acceptable library implementation were proven, it is difficult to justify why the interface itself should not be made portable, especially considering approval for the other four timeouts.

- Rationale for Library Implementation of *pthread\_timedjoin()*

Library implementation of *pthread\_timedjoin()*:

```

6120
6121
6122 /*
6123 * Construct a thread variety entirely from existing functions
6124 * with which a join can be done, allowing the join to time out.
6125 */
6126 #include <pthread.h>
6127 #include <time.h>
6128
6129 struct timed_thread {
6130 pthread_t t;
6131 pthread_mutex_t m;
6132 int exiting;
6133 pthread_cond_t exit_c;
6134 void *(*start_routine)(void *arg);
6135 void *arg;
6136 void *status;
6137 };
6138
6139 typedef struct timed_thread *timed_thread_t;
6140 static pthread_key_t timed_thread_key;
6141 static pthread_once_t timed_thread_once = PTHREAD_ONCE_INIT;
6142
6143 static void timed_thread_init()
6144 {
6145 pthread_key_create(&timed_thread_key, NULL);
6146 }
6147
6148 static void *timed_thread_start_routine(void *args)
6149 /*
6150 * Routine to establish thread-specific data value and run the actual
6151 * thread start routine which was supplied to timed_thread_create().
6152 */
6153 {
6154 timed_thread_t tt = (timed_thread_t) args;
6155
6156 pthread_once(&timed_thread_once, timed_thread_init);
6157 pthread_setspecific(timed_thread_key, (void *)tt);
6158 timed_thread_exit((tt->start_routine)(tt->arg));
6159 }
6160
6161 int timed_thread_create(timed_thread_t ttp, const pthread_attr_t *attr,
6162 void *(*start_routine)(void *), void *arg)
6163 /*
6164 * Allocate a thread which can be used with timed_thread_join().
6165 */
6166 {
6167 timed_thread_t tt;
6168 int result;
6169
6170 tt = (timed_thread_t) malloc(sizeof(struct timed_thread));
6171 pthread_mutex_init(&tt->m, NULL);
6172 tt->exiting = FALSE;
6173 pthread_cond_init(&tt->exit_c, NULL);

```

```

6167 tt->start_routine = start_routine;
6168 tt->arg = arg;
6169 tt->status = NULL;

6170 if ((result = pthread_create(&tt->t, attr,
6171 timed_thread_start_routine, (void *)tt)) != 0) {
6172 free(tt);
6173 return result;
6174 }

6175 pthread_detach(tt->t);
6176 ttp = tt;
6177 return 0;
6178 }

6179 int timed_thread_join(timed_thread_t tt,
6180 struct timespec *timeout,
6181 void **status)
6182 {
6183 int result;

6184 pthread_mutex_lock(&tt->m);
6185 result = 0;
6186 /*
6187 * Wait until the thread announces that it is exiting,
6188 * or until timeout.
6189 */
6190 while (result == 0 && ! tt->exiting) {
6191 result = pthread_cond_timedwait(&tt->exit_c, &tt->m, timeout);
6192 }
6193 pthread_mutex_unlock(&tt->m);
6194 if (result == 0 && tt->exiting) {
6195 *status = tt->status;
6196 free((void *)tt);
6197 return result;
6198 }
6199 return result;
6200 }

6201 void timed_thread_exit(void *status)
6202 {
6203 timed_thread_t tt;
6204 void *specific;

6205 if ((specific=pthread_getspecific(timed_thread_key)) == NULL){
6206 /*
6207 * Handle cases which won't happen with correct usage.
6208 */
6209 pthread_exit(NULL);
6210 }
6211 tt = (timed_thread_t) specific;
6212 pthread_mutex_lock(&tt->m);
6213 /*
6214 * Tell a joiner that we're exiting.
6215 */

```

```

6216 tt->status = status;
6217 tt->exiting = TRUE;
6218 pthread_cond_signal(&tt->exit_c);
6219 pthread_mutex_unlock(&tt->m);
6220 /*
6221 * Call pthread_exit() to call destructors and really
6222 * exit the thread.
6223 */
6224 pthread_exit(NULL);
6225 }

```

6226 The *pthread\_join()* C-language example shown above demonstrates that it is possible, using  
 6227 existing pthread facilities, to construct a variety of thread which allows for joining such a  
 6228 thread, but which allows the join operation to time out. It does this by using a  
 6229 *pthread\_cond\_timedwait()* to wait for the thread to exit. A **timed\_thread\_t** descriptor structure  
 6230 is used to pass parameters from the creating thread to the created thread, and from the  
 6231 exiting thread to the joining thread. This implementation is roughly equivalent to what a  
 6232 normal *pthread\_join()* implementation would do, with the single change being that  
 6233 *pthread\_cond\_timedwait()* is used in place of a simple *pthread\_cond\_wait()*.

6234 Since it is possible to implement such a facility entirely from existing pthread interfaces, and  
 6235 with roughly equal efficiency and complexity to an implementation which would be  
 6236 provided directly by a pthreads implementation, it was the consensus of the working group  
 6237 members that any *pthread\_timedjoin()* facility would be unnecessary, and should not be  
 6238 provided.

#### 6239 • Form of the Timeout Interfaces

6240 The working group considered a number of alternative ways to add timeouts to blocking  
 6241 services. At first, a system interface which would specify a one-shot or persistent timeout to  
 6242 be applied to subsequent blocking services invoked by the calling process or thread was  
 6243 considered because it allowed all blocking services to be timed out in a uniform manner with  
 6244 a single additional interface; this was rather quickly rejected because it could easily result in  
 6245 the wrong services being timed out.

6246 It was suggested that a timeout value might be specified as an attribute of the object  
 6247 (semaphore, mutex, message queue, and so on), but there was no consensus on this, either on  
 6248 a case-by-case basis or for all timeouts.

6249 Looking at the two existing timeouts for blocking services indicates that the working group  
 6250 members favor a separate interface for the timed version of a function. However,  
 6251 *pthread\_cond\_timedwait()* utilizes an absolute timeout value while *sigtimedwait()* uses a  
 6252 relative timeout value. The working group members agreed that relative timeout values are  
 6253 appropriate where the timeout mechanism's primary use was to deal with an unexpected or  
 6254 error situation, but they are inappropriate when the timeout must expire at a particular time,  
 6255 or before a specific deadline. For the timeouts being introduced in IEEE Std 1003.1-2001, the  
 6256 working group considered allowing both relative and absolute timeouts as is done with  
 6257 POSIX.1b timers, but ultimately favored the simpler absolute timeout form.

6258 An absolute time measure can be easily implemented on top of an interface that specifies  
 6259 relative time, by reading the clock, calculating the difference between the current time and  
 6260 the desired wake-up time, and issuing a relative timeout call. But there is a race condition  
 6261 with this approach because the thread could be preempted after reading the clock, but before  
 6262 making the timed-out call; in this case, the thread would be awakened later than it should  
 6263 and, thus, if the wake-up time represented a deadline, it would miss it.

There is also a race condition when trying to build a relative timeout on top of an interface that specifies absolute timeouts. In this case, the clock would have to be read to calculate the absolute wake-up time as the sum of the current time plus the relative timeout interval. In this case, if the thread is preempted after reading the clock but before making the timed-out call, the thread would be awakened earlier than desired.

But the race condition with the absolute timeouts interface is not as bad as the one that happens with the relative timeout interface, because there are simple workarounds. For the absolute timeouts interface, if the timing requirement is a deadline, the deadline can still be met because the thread woke up earlier than the deadline. If the timeout is just used as an error recovery mechanism, the precision of timing is not really important. If the timing requirement is that between actions A and B a minimum interval of time must elapse, the absolute timeout interface can be safely used by reading the clock after action A has been started. It could be argued that, since the call with the absolute timeout is atomic from the application point of view, it is not possible to read the clock after action A, if this action is part of the timed-out call. But looking at the nature of the calls for which timeouts are specified (locking a mutex, waiting for a semaphore, waiting for a message, or waiting until there is space in a message queue), the timeouts that an application would build on these actions would not be triggered by these actions themselves, but by some other external action. For example, if waiting for a message to arrive to a message queue, and waiting for at least 20 milliseconds, this time interval would start to be counted from some event that would trigger both the action that produces the message, as well as the action that waits for the message to arrive, and not by the wait-for-message operation itself. In this case, the workaround proposed above could be used.

For these reasons, the absolute timeout is preferred over the relative timeout interface.

## **B.2.9 Threads**

Threads will normally be more expensive than subroutines (or functions, routines, and so on) if specialized hardware support is not provided. Nevertheless, threads should be sufficiently efficient to encourage their use as a medium to fine-grained structuring mechanism for parallelism in an application. Structuring an application using threads then allows it to take immediate advantage of any underlying parallelism available in the host environment. This means implementors are encouraged to optimize for fast execution at the possible expense of efficient utilization of storage. For example, a common thread creation technique is to cache appropriate thread data structures. That is, rather than releasing system resources, the implementation retains these resources and reuses them when the program next asks to create a new thread. If this reuse of thread resources is to be possible, there has to be very little unique state associated with each thread, because any such state has to be reset when the thread is reused.

### **Thread Creation Attributes**

Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to support probable future standardization in these areas without requiring that the interface itself be changed.

Attributes objects provide clean isolation of the configurable aspects of threads. For example, “stack size” is an important attribute of a thread, but it cannot be expressed portably. When porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects can help by allowing the changes to be isolated in a single place, rather than being spread across every instance of thread creation.

Attributes objects can be used to set up *classes* of threads with similar attributes; for example, “threads with large stacks and high priority” or “threads with minimal stacks”. These classes can be defined in a single place and then referenced wherever threads need to be created. Changes to “class” decisions become straightforward, and detailed analysis of each *pthread\_create()* call is not required.

The attributes objects are defined as opaque types as an aid to extensibility. If these objects had been specified as structures, adding new attributes would force recompilation of all multi-threaded programs when the attributes objects are extended; this might not be possible if different program components were supplied by different vendors.

Additionally, opaque attributes objects present opportunities for improving performance. Argument validity can be checked once when attributes are set, rather than each time a thread is created. Implementations will often need to cache kernel objects that are expensive to create. Opaque attributes objects provide an efficient mechanism to detect when cached objects become invalid due to attribute changes.

Because assignment is not necessarily defined on a given opaque type, implementation-defined default values cannot be defined in a portable way. The solution to this problem is to allow attribute objects to be initialized dynamically by attributes object initialization functions, so that default values can be supplied automatically by the implementation.

The following proposal was provided as a suggested alternative to the supplied attributes:

1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The parameter containing the flags should be an opaque type for extensibility. If no flags are set in the parameter, then the objects are created with default characteristics. An implementation may specify implementation-defined flag values and associated behavior.
2. If further specialization of mutexes and condition variables is necessary, implementations may specify additional procedures that operate on the **pthread\_mutex\_t** and **pthread\_cond\_t** objects (instead of on attributes objects).

The difficulties with this solution are:

1. A bitmask is not opaque if bits have to be set into bit-vector attributes objects using explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**, application programmers need to know the location of each bit. If bits are set or read by encapsulation (that is, *get\*()* or *set\*()* functions), then the bitmask is merely an implementation of attributes objects as currently defined and should not be exposed to the programmer.
2. Many attributes are not Boolean or very small integral values. For example, scheduling policy may be placed in 3 bits or 4 bits, but priority requires 5 bits or more, thereby taking up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the bitmask can only reasonably control whether particular attributes are set or not, and it cannot serve as the repository of the value itself. The value needs to be specified as a function parameter (which is non-extensible), or by setting a structure field (which is non-opaque), or by *get\*()* and *set\*()* functions (making the bitmask a redundant addition to the attributes objects).

Stack size is defined as an optional attribute because the very notion of a stack is inherently machine-dependent. Some implementations may not be able to change the size of the stack, for example, and others may not need to because stack pages may be discontinuous and can be allocated and released on demand.

The attribute mechanism has been designed in large measure for extensibility. Future extensions to the attribute mechanism or to any attributes object defined in IEEE Std 1003.1-2001 have to be done with care so as not to affect binary-compatibility.

Attribute objects, even if allocated by means of dynamic allocation functions such as *malloc()*, may have their size fixed at compile time. This means, for example, a *pthread\_create()* in an implementation with extensions to the **pthread\_attr\_t** cannot look beyond the area that the binary application assumes is valid. This suggests that implementations should maintain a size field in the attributes object, as well as possibly version information, if extensions in different directions (possibly by different vendors) are to be accommodated.

### Thread Implementation Models

There are various thread implementation models. At one end of the spectrum is the “library-thread model”. In such a model, the threads of a process are not visible to the operating system kernel, and the threads are not kernel-scheduled entities. The process is the only kernel-scheduled entity. The process is scheduled onto the processor by the kernel according to the scheduling attributes of the process. The threads are scheduled onto the single kernel-scheduled entity (the process) by the runtime library according to the scheduling attributes of the threads. A problem with this model is that it constrains concurrency. Since there is only one kernel-scheduled entity (namely, the process), only one thread per process can execute at a time. If the thread that is executing blocks on I/O, then the whole process blocks.

At the other end of the spectrum is the “kernel-thread model”. In this model, all threads are visible to the operating system kernel. Thus, all threads are kernel-scheduled entities, and all threads can concurrently execute. The threads are scheduled onto processors by the kernel according to the scheduling attributes of the threads. The drawback to this model is that the creation and management of the threads entails operating system calls, as opposed to subroutine calls, which makes kernel threads heavier weight than library threads.

Hybrids of these two models are common. A hybrid model offers the speed of library threads and the concurrency of kernel threads. In hybrid models, a process has some (relatively small) number of kernel scheduled entities associated with it. It also has a potentially much larger number of library threads associated with it. Some library threads may be bound to kernel-scheduled entities, while the other library threads are multiplexed onto the remaining kernel-scheduled entities. There are two levels of thread scheduling:

1. The runtime library manages the scheduling of (unbound) library threads onto kernel-scheduled entities.
2. The kernel manages the scheduling of kernel-scheduled entities onto processors.

For this reason, a hybrid model is referred to as a two-level threads scheduling model. In this model, the process can have multiple concurrently executing threads; specifically, it can have as many concurrently executing threads as it has kernel-scheduled entities.

### Thread-Specific Data

Many applications require that a certain amount of context be maintained on a per-thread basis across procedure calls. A common example is a multi-threaded library routine that allocates resources from a common pool and maintains an active resource list for each thread. The thread-specific data interface provided to meet these needs may be viewed as a two-dimensional array of values with keys serving as the row index and thread IDs as the column index (although the implementation need not work this way).



- Models

Three possible thread-specific data models were considered:

1. No Explicit Support

A standard thread-specific data interface is not strictly necessary to support applications that require per-thread context. One could, for example, provide a hash function that converted a **pthread\_t** into an integer value that could then be used to index into a global array of per-thread data pointers. This hash function, in conjunction with *pthread\_self()*, would be all the interface required to support a mechanism of this sort. Unfortunately, this technique is cumbersome. It can lead to duplicated code as each set of cooperating modules implements their own per-thread data management schemes.

2. Single (**void \***) Pointer

Another technique would be to provide a single word of per-thread storage and a pair of functions to fetch and store the value of this word. The word could then hold a pointer to a block of per-thread memory. The allocation, partitioning, and general use of this memory would be entirely up to the application. Although this method is not as problematic as technique 1, it suffers from interoperability problems. For example, all modules using the per-thread pointer would have to agree on a common usage protocol.

3. Key/Value Mechanism

This method associates an opaque key (for example, stored in a variable of type **pthread\_key\_t**) with each per-thread datum. These keys play the role of identifiers for per-thread data. This technique is the most generic and avoids the problems noted above, albeit at the cost of some complexity.

The primary advantage of the third model is its information hiding properties. Modules using this model are free to create and use their own key(s) independent of all other such usage, whereas the other models require that all modules that use thread-specific context explicitly cooperate with all other such modules. The data-independence provided by the third model is worth the additional interface.

- Requirements

It is important that it be possible to implement the thread-specific data interface without the use of thread private memory. To do otherwise would increase the weight of each thread, thereby limiting the range of applications for which the threads interfaces provided by IEEE Std 1003.1-2001 is appropriate.

The values that one binds to the key via *pthread\_setspecific()* may, in fact, be pointers to shared storage locations available to all threads. It is only the key/value bindings that are maintained on a per-thread basis, and these can be kept in any portion of the address space that is reserved for use by the calling thread (for example, on the stack). Thus, no per-thread MMU state is required to implement the interface. On the other hand, there is nothing in the interface specification to preclude the use of a per-thread MMU state if it is available (for example, the key values returned by *pthread\_key\_create()* could be thread private memory addresses).

- Standardization Issues

Thread-specific data is a requirement for a usable thread interface. The binding described in this section provides a portable thread-specific data mechanism for languages that do not directly support a thread-specific storage class. A binding to IEEE Std 1003.1-2001 for a

language that does include such a storage class need not provide this specific interface.

If a language were to include the notion of thread-specific storage, it would be desirable (but *not* required) to provide an implementation of the pthreads thread-specific data interface based on the language feature. For example, assume that a compiler for a C-like language supports a *private* storage class that provides thread-specific storage. Something similar to the following macros might be used to effect a compatible implementation:

```
#define pthread_key_t private void *
#define pthread_key_create(key) /* no-op */
#define pthread_setspecific(key,value) (key)=(value)
#define pthread_getspecific(key) (key)
```

**Note:** For the sake of clarity, this example ignores destructor functions. A correct implementation would have to support them.

## Barriers

- Background

Barriers are typically used in parallel DO/FOR loops to ensure that all threads have reached a particular stage in a parallel computation before allowing any to proceed to the next stage. Highly efficient implementation is possible on machines which support a “Fetch and Add” operation as described in the referenced Almasi and Gottlieb (1989).

The use of return value PTHREAD\_BARRIER\_SERIAL\_THREAD is shown in the following example:

```
if ((status=pthread_barrier_wait(&barrier)) ==
 PTHREAD_BARRIER_SERIAL_THREAD) {
 ...serial section
}
else if (status != 0) {
 ...error processing
}
status=pthread_barrier_wait(&barrier);
...
```

This behavior allows a serial section of code to be executed by one thread as soon as all threads reach the first barrier. The second barrier prevents the other threads from proceeding until the serial section being executed by the one thread has completed.

Although barriers can be implemented with mutexes and condition variables, the referenced Almasi and Gottlieb (1989) provides ample illustration that such implementations are significantly less efficient than is possible. While the relative efficiency of barriers may well vary by implementation, it is important that they be recognized in the IEEE Std 1003.1-2001 to facilitate applications portability while providing the necessary freedom to implementors.

- Lack of Timeout Feature

Alternate versions of most blocking routines have been provided to support watchdog timeouts. No alternate interface of this sort has been provided for barrier waits for the following reasons:

- Multiple threads may use different timeout values, some of which may be indefinite. It is not clear which threads should break through the barrier with a timeout error if and when these timeouts expire.

6490                   • The barrier may become unusable once a thread breaks out of a *pthread\_barrier\_wait()*  
 6491                    with a timeout error. There is, in general, no way to guarantee the consistency of a  
 6492                    barrier's internal data structures once a thread has timed out of a *pthread\_barrier\_wait()*.  
 6493                    Even the inclusion of a special barrier reinitialization function would not help much since  
 6494                    it is not clear how this function would affect the behavior of threads that reach the barrier  
 6495                    between the original timeout and the call to the reinitialization function.

## 6496                   **Spin Locks**

### 6497                   • Background

6498                   Spin locks represent an extremely low-level synchronization mechanism suitable primarily  
 6499                   for use on shared memory multi-processors. It is typically an atomically modified Boolean  
 6500                   value that is set to one when the lock is held and to zero when the lock is freed.

6501                   When a caller requests a spin lock that is already held, it typically spins in a loop testing  
 6502                   whether the lock has become available. Such spinning wastes processor cycles so the lock  
 6503                   should only be held for short durations and not across sleep / block operations. Callers should  
 6504                   unlock spin locks before calling sleep operations.

6505                   Spin locks are available on a variety of systems. The functions included in  
 6506                   IEEE Std 1003.1-2001 are an attempt to standardize that existing practice.

### 6507                   • Lack of Timeout Feature

6508                   Alternate versions of most blocking routines have been provided to support watchdog  
 6509                   timeouts. No alternate interface of this sort has been provided for spin locks for the following  
 6510                   reasons:

6511                   • It is impossible to determine appropriate timeout intervals for spin locks in a portable  
 6512                   manner. The amount of time one can expect to spend spin-waiting is inversely  
 6513                   proportional to the degree of parallelism provided by the system.

6514                   It can vary from a few cycles when each competing thread is running on its own  
 6515                   processor, to an indefinite amount of time when all threads are multiplexed on a single  
 6516                   processor (which is why spin locking is not advisable on uniprocessors).

6517                   • When used properly, the amount of time the calling thread spends waiting on a spin lock  
 6518                   should be considerably less than the time required to set up a corresponding watchdog  
 6519                   timer. Since the primary purpose of spin locks is to provide a low-overhead  
 6520                   synchronization mechanism for multi-processors, the overhead of a timeout mechanism  
 6521                   was deemed unacceptable.

6522                   It was also suggested that an additional *count* argument be provided (on the  
 6523                   *pthread\_spin\_lock()* call) in lieu of a true timeout so that a spin lock call could fail gracefully if  
 6524                   it was unable to apply the lock after *count* attempts. This idea was rejected because it is not  
 6525                   existing practice. Furthermore, the same effect can be obtained with *pthread\_spin\_trylock()*,  
 6526                   as illustrated below:

```

6527 int n = MAX_SPIN;
6528 while (--n >= 0)
6529 {
6530 if (!pthread_spin_try_lock(...))
6531 break;
6532 }
6533 if (n >= 0)
6534 {
6535 /* Successfully acquired the lock */
6536 }
6537 else
6538 {
6539 /* Unable to acquire the lock */
6540 }

```

#### 6541 • *process-shared* Attribute

6542 The initialization functions associated with most POSIX synchronization objects (for  
6543 example, mutexes, barriers, and read-write locks) take an attributes object with a *process-*  
6544 *shared* attribute that specifies whether or not the object is to be shared across processes. In the  
6545 draft corresponding to the first balloting round, two separate initialization functions are  
6546 provided for spin locks, however: one for spin locks that were to be shared across processes  
6547 (*spin\_init()*), and one for locks that were only used by multiple threads within a single  
6548 process (*pthread\_spin\_init()*). This was done so as to keep the overhead associated with spin  
6549 waiting to an absolute minimum. However, the balloting group requested that, since the  
6550 overhead associated to a bit check was small, spin locks should be consistent with the rest of  
6551 the synchronization primitives, and thus the *process-shared* attribute was introduced for spin  
6552 locks.

#### 6553 • Spin Locks versus Mutexes

6554 It has been suggested that mutexes are an adequate synchronization mechanism and spin  
6555 locks are not necessary. Locking mechanisms typically must trade off the processor resources  
6556 consumed while setting up to block the thread and the processor resources consumed by the  
6557 thread while it is blocked. Spin locks require very little resources to set up the blocking of a  
6558 thread. Existing practice is to simply loop, repeating the atomic locking operation until the  
6559 lock is available. While the resources consumed to set up blocking of the thread are low, the  
6560 thread continues to consume processor resources while it is waiting.

6561 On the other hand, mutexes may be implemented such that the processor resources  
6562 consumed to block the thread are large relative to a spin lock. After detecting that the mutex  
6563 lock is not available, the thread must alter its scheduling state, add itself to a set of waiting  
6564 threads, and, when the lock becomes available again, undo all of this before taking over  
6565 ownership of the mutex. However, while a thread is blocked by a mutex, no processor  
6566 resources are consumed.

6567 Therefore, spin locks and mutexes may be implemented to have different characteristics.  
6568 Spin locks may have lower overall overhead for very short-term blocking, and mutexes may  
6569 have lower overall overhead when a thread will be blocked for longer periods of time. The  
6570 presence of both interfaces allows implementations with these two different characteristics,  
6571 both of which may be useful to a particular application.

6572 It has also been suggested that applications can build their own spin locks from the  
6573 *pthread\_mutex\_trylock()* function:

```
6574 while (pthread_mutex_trylock(&mutex));
```

6575 The apparent simplicity of this construct is somewhat deceiving, however. While the actual  
6576 wait is quite efficient, various guarantees on the integrity of mutex objects (for example,  
6577 priority inheritance rules) may add overhead to the successful path of the trylock operation  
6578 that is not required of spin locks. One could, of course, add an attribute to the mutex to  
6579 bypass such overhead, but the very act of finding and testing this attribute represents more  
6580 overhead than is found in the typical spin lock.

6581 The need to hold spin lock overhead to an absolute minimum also makes it impossible to  
6582 provide guarantees against starvation similar to those provided for mutexes or read-write  
6583 locks. The overhead required to implement such guarantees (for example, disabling  
6584 preemption before spinning) may well exceed the overhead of the spin wait itself by many  
6585 orders of magnitude. If a “safe” spin wait seems desirable, it can always be provided (albeit  
6586 at some performance cost) via appropriate mutex attributes.

## 6587 XSI Supported Functions

6588 On XSI-conformant systems, the following symbolic constants are always defined:

```
6589 _POSIX_READER_WRITER_LOCKS
6590 _POSIX_THREAD_ATTR_STACKADDR
6591 _POSIX_THREAD_ATTR_STACKSIZE
6592 _POSIX_THREAD_PROCESS_SHARED
6593 _POSIX_THREADS
```

6594 Therefore, the following threads functions are always supported:

|                                           |                                        |
|-------------------------------------------|----------------------------------------|
| 6595 <i>pthread_atfork()</i>              | <i>pthread_key_create()</i>            |
| 6596 <i>pthread_attr_destroy()</i>        | <i>pthread_key_delete()</i>            |
| 6597 <i>pthread_attr_getdetachstate()</i> | <i>pthread_kill()</i>                  |
| 6598 <i>pthread_attr_getguardsize()</i>   | <i>pthread_mutex_destroy()</i>         |
| 6599 <i>pthread_attr_getschedparam()</i>  | <i>pthread_mutex_init()</i>            |
| 6600 <i>pthread_attr_getstack()</i>       | <i>pthread_mutex_lock()</i>            |
| 6601 <i>pthread_attr_getstackaddr()</i>   | <i>pthread_mutex_trylock()</i>         |
| 6602 <i>pthread_attr_getstacksize()</i>   | <i>pthread_mutex_unlock()</i>          |
| 6603 <i>pthread_attr_init()</i>           | <i>pthread_mutexattr_destroy()</i>     |
| 6604 <i>pthread_attr_setdetachstate()</i> | <i>pthread_mutexattr_getpshared()</i>  |
| 6605 <i>pthread_attr_setguardsize()</i>   | <i>pthread_mutexattr_gettype()</i>     |
| 6606 <i>pthread_attr_setschedparam()</i>  | <i>pthread_mutexattr_init()</i>        |
| 6607 <i>pthread_attr_setstack()</i>       | <i>pthread_mutexattr_setpshared()</i>  |
| 6608 <i>pthread_attr_setstackaddr()</i>   | <i>pthread_mutexattr_settype()</i>     |
| 6609 <i>pthread_attr_setstacksize()</i>   | <i>pthread_once()</i>                  |
| 6610 <i>pthread_cancel()</i>              | <i>pthread_rwlock_destroy()</i>        |
| 6611 <i>pthread_cleanup_pop()</i>         | <i>pthread_rwlock_init()</i>           |
| 6612 <i>pthread_cleanup_push()</i>        | <i>pthread_rwlock_rdlock()</i>         |
| 6613 <i>pthread_cond_broadcast()</i>      | <i>pthread_rwlock_tryrdlock()</i>      |
| 6614 <i>pthread_cond_destroy()</i>        | <i>pthread_rwlock_trywrlock()</i>      |
| 6615 <i>pthread_cond_init()</i>           | <i>pthread_rwlock_unlock()</i>         |
| 6616 <i>pthread_cond_signal()</i>         | <i>pthread_rwlock_wrlock()</i>         |
| 6617 <i>pthread_cond_timedwait()</i>      | <i>pthread_rwlockattr_destroy()</i>    |
| 6618 <i>pthread_cond_wait()</i>           | <i>pthread_rwlockattr_getpshared()</i> |
| 6619 <i>pthread_condattr_destroy()</i>    | <i>pthread_rwlockattr_init()</i>       |

|      |                                      |                                        |
|------|--------------------------------------|----------------------------------------|
| 6620 | <i>pthread_condattr_getpshared()</i> | <i>pthread_rwlockattr_setpshared()</i> |
| 6621 | <i>pthread_condattr_init()</i>       | <i>pthread_self()</i>                  |
| 6622 | <i>pthread_condattr_setpshared()</i> | <i>pthread_setcancelstate()</i>        |
| 6623 | <i>pthread_create()</i>              | <i>pthread_setcanceltype()</i>         |
| 6624 | <i>pthread_detach()</i>              | <i>pthread_setconcurrency()</i>        |
| 6625 | <i>pthread_equal()</i>               | <i>pthread_setspecific()</i>           |
| 6626 | <i>pthread_exit()</i>                | <i>pthread_sigmask()</i>               |
| 6627 | <i>pthread_getconcurrency()</i>      | <i>pthread_testcancel()</i>            |
| 6628 | <i>pthread_getspecific()</i>         | <i>sigwait()</i>                       |
| 6629 | <i>pthread_join()</i>                |                                        |

6630 On XSI-conformant systems, the symbolic constant `_POSIX_THREAD_SAFE_FUNCTIONS` is  
 6631 always defined. Therefore, the following functions are always supported:

|      |                           |                           |
|------|---------------------------|---------------------------|
| 6632 | <i>asctime_r()</i>        | <i>getpwuid_r()</i>       |
| 6633 | <i>ctime_r()</i>          | <i>gmtime_r()</i>         |
| 6634 | <i>flockfile()</i>        | <i>localtime_r()</i>      |
| 6635 | <i>ftrylockfile()</i>     | <i>putc_unlocked()</i>    |
| 6636 | <i>funlockfile()</i>      | <i>putchar_unlocked()</i> |
| 6637 | <i>getc_unlocked()</i>    | <i>rand_r()</i>           |
| 6638 | <i>getchar_unlocked()</i> | <i>readdir_r()</i>        |
| 6639 | <i>getgrgid_r()</i>       | <i>strerror_r()</i>       |
| 6640 | <i>getgrnam_r()</i>       | <i>strtok_r()</i>         |
| 6641 | <i>getpwnam_r()</i>       |                           |

6642 The following threads functions are only supported on XSI-conformant systems if the Realtime  
 6643 Threads Option Group is supported :

|      |                                       |                                           |
|------|---------------------------------------|-------------------------------------------|
| 6644 | <i>pthread_attr_getinheritsched()</i> | <i>pthread_mutex_getprioceiling()</i>     |
| 6645 | <i>pthread_attr_getschedpolicy()</i>  | <i>pthread_mutex_setprioceiling()</i>     |
| 6646 | <i>pthread_attr_getscope()</i>        | <i>pthread_mutexattr_getprioceiling()</i> |
| 6647 | <i>pthread_attr_setinheritsched()</i> | <i>pthread_mutexattr_getprotocol()</i>    |
| 6648 | <i>pthread_attr_setschedpolicy()</i>  | <i>pthread_mutexattr_setprioceiling()</i> |
| 6649 | <i>pthread_attr_setscope()</i>        | <i>pthread_mutexattr_setprotocol()</i>    |
| 6650 | <i>pthread_getschedparam()</i>        | <i>pthread_setschedparam()</i>            |

## 6651 XSI Threads Extensions

6652 The following XSI extensions to POSIX.1c are now supported in IEEE Std 1003.1-2001 as part of  
 6653 the alignment with the Single UNIX Specification:

- 6654 • Extended mutex attribute types
- 6655 • Read-write locks and attributes (also introduced by the IEEE Std 1003.1j-2000 amendment)
- 6656 • Thread concurrency level
- 6657 • Thread stack guard size
- 6658 • Parallel I/O

6659 A total of 19 new functions were added.

6660 These extensions carefully follow the threads programming model specified in POSIX.1c. As  
 6661 with POSIX.1c, all the new functions return zero if successful; otherwise, an error number is

returned to indicate the error.

The concept of attribute objects was introduced in POSIX.1c to allow implementations to extend IEEE Std 1003.1-2001 without changing the existing interfaces. Attribute objects were defined for threads, mutexes, and condition variables. Attributes objects are defined as implementation-defined opaque types to aid extensibility, and functions are defined to allow attributes to be set or retrieved. This model has been followed when adding the new type attribute of **pthread\_mutexattr\_t** or the new read-write lock attributes object **pthread\_rwlockattr\_t**.

- Extended Mutex Attributes

POSIX.1c defines a mutex attributes object as an implementation-defined opaque object of type **pthread\_mutexattr\_t**, and specifies a number of attributes which this object must have and a number of functions which manipulate these attributes. These attributes include *detachstate*, *inheritsched*, *schedparm*, *schedpolicy*, *contentionscope*, *stackaddr*, and *stacksize*.

The System Interfaces volume of IEEE Std 1003.1-2001 specifies another mutex attribute called *type*. The *type* attribute allows applications to specify the behavior of mutex locking operations in situations where POSIX.1c behavior is undefined. The OSF DCE threads implementation, based on Draft 4 of POSIX.1c, specified a similar attribute. Note that the names of the attributes have changed somewhat from the OSF DCE threads implementation.

The System Interfaces volume of IEEE Std 1003.1-2001 also extends the specification of the following POSIX.1c functions which manipulate mutexes:

```
pthread_mutex_lock()
pthread_mutex_trylock()
pthread_mutex_unlock()
```

to take account of the new mutex attribute type and to specify behavior which was declared as undefined in POSIX.1c. How a calling thread acquires or releases a mutex now depends upon the mutex *type* attribute.

The *type* attribute can have the following values:

#### PTHREAD\_MUTEX\_NORMAL

Basic mutex with no specific error checking built in. Does not report a deadlock error.

#### PTHREAD\_MUTEX\_RECURSIVE

Allows any thread to recursively lock a mutex. The mutex must be unlocked an equal number of times to release the mutex.

#### PTHREAD\_MUTEX\_ERRORCHECK

Detects and reports simple usage errors; that is, an attempt to unlock a mutex that is not locked by the calling thread or that is not locked at all, or an attempt to relock a mutex the thread already owns.

#### PTHREAD\_MUTEX\_DEFAULT

The default mutex type. May be mapped to any of the above mutex types or may be an implementation-defined type.

*Normal* mutexes do not detect deadlock conditions; for example, a thread will hang if it tries to relock a normal mutex that it already owns. Attempting to unlock a mutex locked by another thread, or unlocking an unlocked mutex, results in undefined behavior. Normal mutexes will usually be the fastest type of mutex available on a platform but provide the least error checking.

*Recursive* mutexes are useful for converting old code where it is difficult to establish clear boundaries of synchronization. A thread can relock a recursive mutex without first unlocking

it. The relocking deadlock which can occur with normal mutexes cannot occur with this type of mutex. However, multiple locks of a recursive mutex require the same number of unlocks to release the mutex before another thread can acquire the mutex. Furthermore, this type of mutex maintains the concept of an owner. Thus, a thread attempting to unlock a recursive mutex which another thread has locked returns with an error. A thread attempting to unlock a recursive mutex that is not locked returns with an error. Never use a recursive mutex with condition variables because the implicit unlock performed by *pthread\_cond\_wait()* or *pthread\_cond\_timedwait()* will not actually release the mutex if it had been locked multiple times.

*Errorcheck* mutexes provide error checking and are useful primarily as a debugging aid. A thread attempting to relock an errorcheck mutex without first unlocking it returns with an error. Again, this type of mutex maintains the concept of an owner. Thus, a thread attempting to unlock an errorcheck mutex which another thread has locked returns with an error. A thread attempting to unlock an errorcheck mutex that is not locked also returns with an error. It should be noted that errorcheck mutexes will almost always be much slower than normal mutexes due to the extra state checks performed.

The default mutex type provides implementation-defined error checking. The default mutex may be mapped to one of the other defined types or may be something entirely different. This enables each vendor to provide the mutex semantics which the vendor feels will be most useful to their target users. Most vendors will probably choose to make normal mutexes the default so as to give applications the benefit of the fastest type of mutexes available on their platform. Check your implementation's documentation.

An application developer can use any of the mutex types almost interchangeably as long as the application does not depend upon the implementation detecting (or failing to detect) any particular errors. Note that a recursive mutex can be used with condition variable waits as long as the application never recursively locks the mutex.

Two functions are provided for manipulating the *type* attribute of a mutex attributes object. This attribute is set or returned in the *type* parameter of these functions. The *pthread\_mutexattr\_settype()* function is used to set a specific type value while *pthread\_mutexattr\_gettype()* is used to return the type of the mutex. Setting the *type* attribute of a mutex attributes object affects only mutexes initialized using that mutex attributes object. Changing the *type* attribute does not affect mutexes previously initialized using that mutex attributes object.

- Read-Write Locks and Attributes

The read-write locks introduced have been harmonized with those in IEEE Std 1003.1j-2000; see also Section B.2.9.6 (on page 175).

Read-write locks (also known as reader-writer locks) allow a thread to exclusively lock some shared data while updating that data, or allow any number of threads to have simultaneous read-only access to the data.

Unlike a mutex, a read-write lock distinguishes between reading data and writing data. A mutex excludes all other threads. A read-write lock allows other threads access to the data, providing no thread is modifying the data. Thus, a read-write lock is less primitive than either a mutex-condition variable pair or a semaphore.

Application developers should consider using a read-write lock rather than a mutex to protect data that is frequently referenced but seldom modified. Most threads (readers) will be able to read the data without waiting and will only have to block when some other thread (a writer) is in the process of modifying the data. Conversely a thread that wants to change the data is forced to wait until there are no readers. This type of lock is often used to facilitate



parallel access to data on multi-processor platforms or to avoid context switches on single processor platforms where multiple threads access the same data.

If a read-write lock becomes unlocked and there are multiple threads waiting to acquire the write lock, the implementation's scheduling policy determines which thread acquires the read-write lock for writing. If there are multiple threads blocked on a read-write lock for both read locks and write locks, it is unspecified whether the readers or a writer acquire the lock first. However, for performance reasons, implementations often favor writers over readers to avoid potential writer starvation.

A read-write lock object is an implementation-defined opaque object of type **pthread\_rwlock\_t** as defined in **<pthread.h>**. There are two different sorts of locks associated with a read-write lock: a read lock and a write lock.

The *pthread\_rwlockattr\_init()* function initializes a read-write lock attributes object with the default value for all the attributes defined in the implementation. After a read-write lock attributes object has been used to initialize one or more read-write locks, changes to the read-write lock attributes object, including destruction, do not affect previously initialized read-write locks.

Implementations must provide at least the read-write lock attribute *process-shared*. This attribute can have the following values:

**PTHREAD\_PROCESS\_SHARED**  
Any thread of any process that has access to the memory where the read-write lock resides can manipulate the read-write lock.

**PTHREAD\_PROCESS\_PRIVATE**  
Only threads created within the same process as the thread that initialized the read-write lock can manipulate the read-write lock. This is the default value.

The *pthread\_rwlockattr\_setpshared()* function is used to set the *process-shared* attribute of an initialized read-write lock attributes object while the function *pthread\_rwlockattr\_getpshared()* obtains the current value of the *process-shared* attribute.

A read-write lock attributes object is destroyed using the *pthread\_rwlockattr\_destroy()* function. The effect of subsequent use of the read-write lock attributes object is undefined.

A thread creates a read-write lock using the *pthread\_rwlock\_init()* function. The attributes of the read-write lock can be specified by the application developer; otherwise, the default implementation-defined read-write lock attributes are used if the pointer to the read-write lock attributes object is NULL. In cases where the default attributes are appropriate, the **PTHREAD\_RWLOCK\_INITIALIZER** macro can be used to initialize statically allocated read-write locks.

A thread which wants to apply a read lock to the read-write lock can use either *pthread\_rwlock\_rdlock()* or *pthread\_rwlock\_tryrdlock()*. If *pthread\_rwlock\_rdlock()* is used, the thread acquires a read lock if a writer does not hold the write lock and there are no writers blocked on the write lock. If a read lock is not acquired, the calling thread blocks until it can acquire a lock. However, if *pthread\_rwlock\_tryrdlock()* is used, the function returns immediately with the error [EBUSY] if any thread holds a write lock or there are blocked writers waiting for the write lock.

A thread which wants to apply a write lock to the read-write lock can use either of two functions: *pthread\_rwlock\_wrlock()* or *pthread\_rwlock\_trywrlock()*. If *pthread\_rwlock\_wrlock()* is used, the thread acquires the write lock if no other reader or writer threads hold the read-write lock. If the write lock is not acquired, the thread blocks until it can acquire the write lock. However, if *pthread\_rwlock\_trywrlock()* is used, the function returns immediately with

the error [EBUSY] if any thread is holding either a read or a write lock.

The `pthread_rwlock_unlock()` function is used to unlock a read-write lock object held by the calling thread. Results are undefined if the read-write lock is not held by the calling thread. If there are other read locks currently held on the read-write lock object, the read-write lock object remains in the read locked state but without the current thread as one of its owners. If this function releases the last read lock for this read-write lock object, the read-write lock object is put in the unlocked read state. If this function is called to release a write lock for this read-write lock object, the read-write lock object is put in the unlocked state.

- Thread Concurrency Level

On threads implementations that multiplex user threads onto a smaller set of kernel execution entities, the system attempts to create a reasonable number of kernel execution entities for the application upon application startup.

On some implementations, these kernel entities are retained by user threads that block in the kernel. Other implementations do not *timeslice* user threads so that multiple compute-bound user threads can share a kernel thread. On such implementations, some applications may use up all the available kernel execution entities before their user-space threads are used up. The process may be left with user threads capable of doing work for the application but with no way to schedule them.

The `pthread_setconcurrency()` function enables an application to request more kernel entities; that is, specify a desired concurrency level. However, this function merely provides a hint to the implementation. The implementation is free to ignore this request or to provide some other number of kernel entities. If an implementation does not multiplex user threads onto a smaller number of kernel execution entities, the `pthread_setconcurrency()` function has no effect.

The `pthread_setconcurrency()` function may also have an effect on implementations where the kernel mode and user mode schedulers cooperate to ensure that ready user threads are not prevented from running by other threads blocked in the kernel.

The `pthread_getconcurrency()` function always returns the value set by a previous call to `pthread_setconcurrency()`. However, if `pthread_setconcurrency()` was not previously called, this function returns zero to indicate that the threads implementation is maintaining the concurrency level.

- Thread Stack Guard Size

DCE threads introduced the concept of a “thread stack guard size”. Most thread implementations add a region of protected memory to a thread’s stack, commonly known as a “guard region”, as a safety measure to prevent stack pointer overflow in one thread from corrupting the contents of another thread’s stack. The default size of the guard regions attribute is {PAGESIZE} bytes and is implementation-defined.

Some application developers may wish to change the stack guard size. When an application creates a large number of threads, the extra page allocated for each stack may strain system resources. In addition to the extra page of memory, the kernel’s memory manager has to keep track of the different protections on adjoining pages. When this is a problem, the application developer may request a guard size of 0 bytes to conserve system resources by eliminating stack overflow protection.

Conversely an application that allocates large data structures such as arrays on the stack may wish to increase the default guard size in order to detect stack overflow. If a thread allocates two pages for a data array, a single guard page provides little protection against thread stack overflows since the thread can corrupt adjoining memory beyond the guard page.

6849 The System Interfaces volume of IEEE Std 1003.1-2001 defines a new attribute of a thread  
 6850 attributes object; that is, the *guardsize* attribute which allows applications to specify the size  
 6851 of the guard region of a thread's stack.

6852 Two functions are provided for manipulating a thread's stack guard size. The  
 6853 *pthread\_attr\_setguardsize()* function sets the thread *guardsize* attribute, and the  
 6854 *pthread\_attr\_getguardsize()* function retrieves the current value.

6855 An implementation may round up the requested guard size to a multiple of the configurable  
 6856 system variable {PAGESIZE}. In this case, *pthread\_attr\_getguardsize()* returns the guard size  
 6857 specified by the previous *pthread\_attr\_setguardsize()* function call and not the rounded up  
 6858 value.

6859 If an application is managing its own thread stacks using the *stackaddr* attribute, the *guardsize*  
 6860 attribute is ignored and no stack overflow protection is provided. In this case, it is the  
 6861 responsibility of the application to manage stack overflow along with stack allocation.

#### 6862 • Parallel I/O

6863 Suppose two or more threads independently issue read requests on the same file. To read  
 6864 specific data from a file, a thread must first call *lseek()* to seek to the proper offset in the file,  
 6865 and then call *read()* to retrieve the required data. If more than one thread does this at the  
 6866 same time, the first thread may complete its seek call, but before it gets a chance to issue its  
 6867 read call a second thread may complete its seek call, resulting in the first thread accessing  
 6868 incorrect data when it issues its read call. One workaround is to lock the file descriptor while  
 6869 seeking and reading or writing, but this reduces parallelism and adds overhead.

6870 Instead, the System Interfaces volume of IEEE Std 1003.1-2001 provides two functions to  
 6871 make seek/read and seek/write operations atomic. The file descriptor's current offset is  
 6872 unchanged, thus allowing multiple read and write operations to proceed in parallel. This  
 6873 improves the I/O performance of threaded applications. The *pread()* function is used to do  
 6874 an atomic read of data from a file into a buffer. Conversely, the *pwrite()* function does an  
 6875 atomic write of data from a buffer to a file.

#### 6876 B.2.9.1 Thread-Safety

6877 All functions required by IEEE Std 1003.1-2001 need to be thread-safe. Implementations have to  
 6878 provide internal synchronization when necessary in order to achieve this goal. In certain  
 6879 cases—for example, most floating-point implementations—context switch code may have to  
 6880 manage the writable shared state.

6881 While a read from a pipe of {PIPE\_MAX}\*2 bytes may not generate a single atomic and thread-  
 6882 safe stream of bytes, it should generate “several” (individually atomic) thread-safe streams of  
 6883 bytes. Similarly, while reading from a terminal device may not generate a single atomic and  
 6884 thread-safe stream of bytes, it should generate some finite number of (individually atomic) and  
 6885 thread-safe streams of bytes. That is, concurrent calls to read for a pipe, FIFO, or terminal device  
 6886 are not allowed to result in corrupting the stream of bytes or other internal data. However,  
 6887 *read()*, in these cases, is not required to return a single contiguous and atomic stream of bytes.

6888 It is not required that all functions provided by IEEE Std 1003.1-2001 be either async-cancel-safe  
 6889 or async-signal-safe.

6890 As it turns out, some functions are inherently not thread-safe; that is, their interface  
 6891 specifications preclude reentrancy. For example, some functions (such as *asctime()*) return a  
 6892 pointer to a result stored in memory space allocated by the function on a per-process basis. Such  
 6893 a function is not thread-safe, because its result can be overwritten by successive invocations.  
 6894 Other functions, while not inherently non-thread-safe, may be implemented in ways that lead to

them not being thread-safe. For example, some functions (such as *rand()*) store state information (such as a seed value, which survives multiple function invocations) in memory space allocated by the function on a per-process basis. The implementation of such a function is not thread-safe if the implementation fails to synchronize invocations of the function and thus fails to protect the state information. The problem is that when the state information is not protected, concurrent invocations can interfere with one another (for example, applications using *rand()* may see the same seed value).

#### *Thread-Safety and Locking of Existing Functions*

Originally, POSIX.1 was not designed to work in a multi-threaded environment, and some implementations of some existing functions will not work properly when executed concurrently. To provide routines that will work correctly in an environment with threads (“thread-safe”), two problems need to be solved:

1. Routines that maintain or return pointers to static areas internal to the routine (which may now be shared) need to be modified. The routines *ttyname()* and *localtime()* are examples.
2. Routines that access data space shared by more than one thread need to be modified. The *malloc()* function and the *stdio* family routines are examples.

There are a variety of constraints on these changes. The first is compatibility with the existing versions of these functions—non-thread-safe functions will continue to be in use for some time, as the original interfaces are used by existing code. Another is that the new thread-safe versions of these functions represent as small a change as possible over the familiar interfaces provided by the existing non-thread-safe versions. The new interfaces should be independent of any particular threads implementation. In particular, they should be thread-safe without depending on explicit thread-specific memory. Finally, there should be minimal performance penalty due to the changes made to the functions.

It is intended that the list of functions from POSIX.1 that cannot be made thread-safe and for which corrected versions are provided be complete.

#### *Thread-Safety and Locking Solutions*

Many of the POSIX.1 functions were thread-safe and did not change at all. However, some functions (for example, the math functions typically found in **libm**) are not thread-safe because of writable shared global state. For instance, in IEEE Std 754-1985 floating-point implementations, the computation modes and flags are global and shared.

Some functions are not thread-safe because a particular implementation is not reentrant, typically because of a non-essential use of static storage. These require only a new implementation.

Thread-safe libraries are useful in a wide range of parallel (and asynchronous) programming environments, not just within pthreads. In order to be used outside the context of pthreads, however, such libraries still have to use some synchronization method. These could either be independent of the pthread synchronization operations, or they could be a subset of the pthread interfaces. Either method results in thread-safe library implementations that can be used without the rest of pthreads.

Some functions, such as the *stdio* family interface and dynamic memory allocation functions such as *malloc()*, are inter-dependent routines that share resources (for example, buffers) across related calls. These require synchronization to work correctly, but they do not require any change to their external (user-visible) interfaces.

In some cases, such as *getc()* and *putc()*, adding synchronization is likely to create an unacceptable performance impact. In this case, slower thread-safe synchronized functions are to

be provided, but the original, faster (but unsafe) functions (which may be implemented as macros) are retained under new names. Some additional special-purpose synchronization facilities are necessary for these macros to be usable in multi-threaded programs. This also requires changes in `<stdio.h>`.

The other common reason that functions are unsafe is that they return a pointer to static storage, making the functions non-thread-safe. This has to be changed, and there are three natural choices:

1. Return a pointer to thread-specific storage

This could incur a severe performance penalty on those architectures with a costly implementation of the thread-specific data interface.

A variation on this technique is to use `malloc()` to allocate storage for the function output and return a pointer to this storage. This technique may also have an undesirable performance impact, however, and a simplistic implementation requires that the user program explicitly free the storage object when it is no longer needed. This technique is used by some existing POSIX.1 functions. With careful implementation for infrequently used functions, there may be little or no performance or storage penalty, and the maintenance of already-standardized interfaces is a significant benefit.

2. Return the actual value computed by the function

This technique can only be used with functions that return pointers to structures—routines that return character strings would have to wrap their output in an enclosing structure in order to return the output on the stack. There is also a negative performance impact inherent in this solution in that the output value has to be copied twice before it can be used by the calling function: once from the called routine's local buffers to the top of the stack, then from the top of the stack to the assignment target. Finally, many older compilers cannot support this technique due to a historical tendency to use internal static buffers to deliver the results of structure-valued functions.

3. Have the caller pass the address of a buffer to contain the computed value

The only disadvantage of this approach is that extra arguments have to be provided by the calling program. It represents the most efficient solution to the problem, however, and, unlike the `malloc()` technique, it is semantically clear.

There are some routines (often groups of related routines) whose interfaces are inherently non-thread-safe because they communicate across multiple function invocations by means of static memory locations. The solution is to redesign the calls so that they are thread-safe, typically by passing the needed data as extra parameters. Unfortunately, this may require major changes to the interface as well.

A floating-point implementation using IEEE Std 754-1985 is a case in point. A less problematic example is the `rand48` family of pseudo-random number generators. The functions `getgrgid()`, `getgrnam()`, `getpwnam()`, and `getpwuid()` are another such case.

The problems with `errno` are discussed in **Alternative Solutions for Per-Thread `errno`** (on page 92).

Some functions can be thread-safe or not, depending on their arguments. These include the `tmpnam()` and `ctermid()` functions. These functions have pointers to character strings as arguments. If the pointers are not NULL, the functions store their results in the character string; however, if the pointers are NULL, the functions store their results in an area that may be static and thus subject to overwriting by successive calls. These should only be called by multi-thread applications when their arguments are non-NULL.

*Asynchronous Safety and Thread-Safety*

A floating-point implementation has many modes that effect rounding and other aspects of computation. Functions in some math library implementations may change the computation modes for the duration of a function call. If such a function call is interrupted by a signal or cancellation, the floating-point state is not required to be protected.

There is a significant cost to make floating-point operations async-cancel-safe or async-signal-safe; accordingly, neither form of async safety is required.

*Functions Returning Pointers to Static Storage*

For those functions that are not thread-safe because they return values in fixed size statically allocated structures, alternate “\_r” forms are provided that pass a pointer to an explicit result structure. Those that return pointers into library-allocated buffers have forms provided with explicit buffer and length parameters.

For functions that return pointers to library-allocated buffers, it makes sense to provide “\_r” versions that allow the application control over allocation of the storage in which results are returned. This allows the state used by these functions to be managed on an application-specific basis, supporting per-thread, per-process, or other application-specific sharing relationships.

Early proposals had provided “\_r” versions for functions that returned pointers to variable-size buffers without providing a means for determining the required buffer size. This would have made using such functions exceedingly clumsy, potentially requiring iteratively calling them with increasingly larger guesses for the amount of storage required. Hence, *sysconf()* variables have been provided for such functions that return the maximum required buffer size.

Thus, the rule that has been followed by IEEE Std 1003.1-2001 when adapting single-threaded non-thread-safe functions is as follows: all functions returning pointers to library-allocated storage should have “\_r” versions provided, allowing the application control over the storage allocation. Those with variable-sized return values accept both a buffer address and a length parameter. The *sysconf()* variables are provided to supply the appropriate buffer sizes when required. Implementors are encouraged to apply the same rule when adapting their own existing functions to a pthreads environment.

*B.2.9.2 Thread IDs*

Separate applications should communicate through well-defined interfaces and should not depend on each other's implementation. For example, if a programmer decides to rewrite the *sort* utility using multiple threads, it should be easy to do this so that the interface to the *sort* utility does not change. Consider that if the user causes SIGINT to be generated while the *sort* utility is running, keeping the same interface means that the entire *sort* utility is killed, not just one of its threads. As another example, consider a realtime application that manages a reactor. Such an application may wish to allow other applications to control the priority at which it watches the control rods. One technique to accomplish this is to write the ID of the thread watching the control rods into a file and allow other programs to change the priority of that thread as they see fit. A simpler technique is to have the reactor process accept IPCs (Interprocess Communication messages) from other processes, telling it at a semantic level what priority the program should assign to watching the control rods. This allows the programmer greater flexibility in the implementation. For example, the programmer can change the implementation from having one thread per rod to having one thread watching all of the rods without changing the interface. Having threads live inside the process means that the implementation of a process is invisible to outside processes (excepting debuggers and system management tools).

Threads do not provide a protection boundary. Every thread model allows threads to share memory with other threads and encourages this sharing to be widespread. This means that one

7034 thread can wipe out memory that is needed for the correct functioning of other threads that are  
7035 sharing its memory. Consequently, providing each thread with its own user and/or group IDs  
7036 would not provide a protection boundary between threads sharing memory.

### 7037 B.2.9.3 Thread Mutexes

7038 There is no additional rationale provided for this section.

### 7039 B.2.9.4 Thread Scheduling

#### 7040 • Scheduling Implementation Models

7041 The following scheduling implementation models are presented in terms of threads and  
7042 “kernel entities”. This is to simplify exposition of the models, and it does not imply that an  
7043 implementation actually has an identifiable “kernel entity”.

7044 A kernel entity is not defined beyond the fact that it has scheduling attributes that are used to  
7045 resolve contention with other kernel entities for execution resources. A kernel entity may be  
7046 thought of as an envelope that holds a thread or a separate kernel thread. It is not a  
7047 conventional process, although it shares with the process the attribute that it has a single  
7048 thread of control; it does not necessarily imply an address space, open files, and so on. It is  
7049 better thought of as a primitive facility upon which conventional processes and threads may  
7050 be constructed.

#### 7051 — System Thread Scheduling Model

7052 This model consists of one thread per kernel entity. The kernel entity is solely responsible  
7053 for scheduling thread execution on one or more processors. This model schedules all  
7054 threads against all other threads in the system using the scheduling attributes of the  
7055 thread.

#### 7056 — Process Scheduling Model

7057 A generalized process scheduling model consists of two levels of scheduling. A threads  
7058 library creates a pool of kernel entities, as required, and schedules threads to run on them  
7059 using the scheduling attributes of the threads. Typically, the size of the pool is a function  
7060 of the simultaneously runnable threads, not the total number of threads. The kernel then  
7061 schedules the kernel entities onto processors according to their scheduling attributes,  
7062 which are managed by the threads library. This set model potentially allows a wide range  
7063 of mappings between threads and kernel entities.

#### 7064 • System and Process Scheduling Model Performance

7065 There are a number of important implications on the performance of applications using these  
7066 scheduling models. The process scheduling model potentially provides lower overhead for  
7067 making scheduling decisions, since there is no need to access kernel-level information or  
7068 functions and the set of schedulable entities is smaller (only the threads within the process).

7069 On the other hand, since the kernel is also making scheduling decisions regarding the system  
7070 resources under its control (for example, CPU(s), I/O devices, memory), decisions that do  
7071 not take thread scheduling parameters into account can result in unspecified delays for  
7072 realtime application threads, causing them to miss maximum response time limits.

#### 7073 • Rate Monotonic Scheduling

7074 Rate monotonic scheduling was considered, but rejected for standardization in the context of  
7075 pthreads. A sporadic server policy is included.

- Scheduling Options

In IEEE Std 1003.1-2001, the basic thread scheduling functions are defined under the Threads option, so that they are required of all threads implementations. However, there are no specific scheduling policies required by this option to allow for conforming thread implementations that are not targeted to realtime applications.

Specific standard scheduling policies are defined to be under the Thread Execution Scheduling option, and they are specifically designed to support realtime applications by providing predictable resource-sharing sequences. The name of this option was chosen to emphasize that this functionality is defined as appropriate for realtime applications that require simple priority-based scheduling.

It is recognized that these policies are not necessarily satisfactory for some multi-processor implementations, and work is ongoing to address a wider range of scheduling behaviors. The interfaces have been chosen to create abundant opportunity for future scheduling policies to be implemented and standardized based on this interface. In order to standardize a new scheduling policy, all that is required (from the standpoint of thread scheduling attributes) is to define a new policy name, new members of the thread attributes object, and functions to set these members when the scheduling policy is equal to the new value.

### **Scheduling Contention Scope**

In order to accommodate the requirement for realtime response, each thread has a scheduling contention scope attribute. Threads with a system scheduling contention scope have to be scheduled with respect to all other threads in the system. These threads are usually bound to a single kernel entity that reflects their scheduling attributes and are directly scheduled by the kernel.

Threads with a process scheduling contention scope need be scheduled only with respect to the other threads in the process. These threads may be scheduled within the process onto a pool of kernel entities. The implementation is also free to bind these threads directly to kernel entities and let them be scheduled by the kernel. Process scheduling contention scope allows the implementation the most flexibility and is the default if both contention scopes are supported and none is specified.

Thus, the choice by implementors to provide one or the other (or both) of these scheduling models is driven by the need of their supported application domains for worst-case (that is, realtime) response, or average-case (non-realtime) response.

### **Scheduling Allocation Domain**

The SCHED\_FIFO and SCHED\_RR scheduling policies take on different characteristics on a multi-processor. Other scheduling policies are also subject to changed behavior when executed on a multi-processor. The concept of scheduling allocation domain determines the set of processors on which the threads of an application may run. By considering the application's processor scheduling allocation domain for its threads, scheduling policies can be defined in terms of their behavior for varying processor scheduling allocation domain values. It is conceivable that not all scheduling allocation domain sizes make sense for all scheduling policies on all implementations. The concept of scheduling allocation domain, however, is a useful tool for the description of multi-processor scheduling policies.

The “process control” approach to scheduling obtains significant performance advantages from dynamic scheduling allocation domain sizes when it is applicable.

Non-Uniform Memory Access (NUMA) multi-processors may use a system scheduling structure that involves reassignment of threads among scheduling allocation domains. In NUMA



machines, a natural model of scheduling is to match scheduling allocation domains to clusters of processors. Load balancing in such an environment requires changing the scheduling allocation domain to which a thread is assigned.

### **Scheduling Documentation**

Implementation-provided scheduling policies need to be completely documented in order to be useful. This documentation includes a description of the attributes required for the policy, the scheduling interaction of threads running under this policy and all other supported policies, and the effects of all possible values for processor scheduling allocation domain. Note that for the implementor wishing to be minimally-compliant, it is (minimally) acceptable to define the behavior as undefined.

### **Scheduling Contention Scope Attribute**

The scheduling contention scope defines how threads compete for resources. Within IEEE Std 1003.1-2001, scheduling contention scope is used to describe only how threads are scheduled in relation to one another in the system. That is, either they are scheduled against all other threads in the system (“system scope”) or only against those threads in the process (“process scope”). In fact, scheduling contention scope may apply to additional resources, including virtual timers and profiling, which are not currently considered by IEEE Std 1003.1-2001.

### **Mixed Scopes**

If only one scheduling contention scope is supported, the scheduling decision is straightforward. To perform the processor scheduling decision in a mixed scope environment, it is necessary to map the scheduling attributes of the thread with process-wide contention scope to the same attribute space as the thread with system-wide contention scope.

Since a conforming implementation has to support one and may support both scopes, it is useful to discuss the effects of such choices with respect to example applications. If an implementation supports both scopes, mixing scopes provides a means of better managing system-level (that is, kernel-level) and library-level resources. In general, threads with system scope will require the resources of a separate kernel entity in order to guarantee the scheduling semantics. On the other hand, threads with process scope can share the resources of a kernel entity while maintaining the scheduling semantics.

The application is free to create threads with dedicated kernel resources, and other threads that multiplex kernel resources. Consider the example of a window server. The server allocates two threads per widget: one thread manages the widget user interface (including drawing), while the other thread takes any required application action. This allows the widget to be “active” while the application is computing. A screen image may be built from thousands of widgets. If each of these threads had been created with system scope, then most of the kernel-level resources might be wasted, since only a few widgets are active at any one time. In addition, mixed scope is particularly useful in a window server where one thread with high priority and system scope handles the mouse so that it tracks well. As another example, consider a database server. For each of the hundreds or thousands of clients supported by a large server, an equivalent number of threads will have to be created. If each of these threads were system scope, the consequences would be the same as for the window server example above. However, the server could be constructed so that actual retrieval of data is done by several dedicated threads. Dedicated threads that do work for all clients frequently justify the added expense of system scope. If it were not permissible to mix system and process threads in the same process, this type of solution would not be possible.

**Dynamic Thread Scheduling Parameters Access**

In many time-constrained applications, there is no need to change the scheduling attributes dynamically during thread or process execution, since the general use of these attributes is to reflect directly the time constraints of the application. Since these time constraints are generally imposed to meet higher-level system requirements, such as accuracy or availability, they frequently should remain unchanged during application execution.

However, there are important situations in which the scheduling attributes should be changed. Generally, this will occur when external environmental conditions exist in which the time constraints change. Consider, for example, a space vehicle major mode change, such as the change from ascent to descent mode, or the change from the space environment to the atmospheric environment. In such cases, the frequency with which many of the sensors or actuators need to be read or written will change, which will necessitate a priority change. In other cases, even the existence of a time constraint might be temporary, necessitating not just a priority change, but also a policy change for ongoing threads or processes. For this reason, it is critical that the interface should provide functions to change the scheduling parameters dynamically, but, as with many of the other realtime functions, it is important that applications use them properly to avoid the possibility of unnecessarily degrading performance.

In providing functions for dynamically changing the scheduling behavior of threads, there were two options: provide functions to get and set the individual scheduling parameters of threads, or provide a single interface to get and set all the scheduling parameters for a given thread simultaneously. Both approaches have merit. Access functions for individual parameters allow simpler control of thread scheduling for simple thread scheduling parameters. However, a single function for setting all the parameters for a given scheduling policy is required when first setting that scheduling policy. Since the single all-encompassing functions are required, it was decided to leave the interface as minimal as possible. Note that simpler functions (such as *pthread\_setprio()* for threads running under the priority-based schedulers) can be easily defined in terms of the all-encompassing functions.

If the *pthread\_setschedparam()* function executes successfully, it will have set all of the scheduling parameter values indicated in *param*; otherwise, none of the scheduling parameters will have been modified. This is necessary to ensure that the scheduling of this and all other threads continues to be consistent in the presence of an erroneous scheduling parameter.

The [EPERM] error value is included in the list of possible *pthread\_setschedparam()* error returns as a reflection of the fact that the ability to change scheduling parameters increases risks to the implementation and application performance if the scheduling parameters are changed improperly. For this reason, and based on some existing practice, it was felt that some implementations would probably choose to define specific permissions for changing either a thread's own or another thread's scheduling parameters. IEEE Std 1003.1-2001 does not include portable methods for setting or retrieving permissions, so any such use of permissions is completely unspecified.

**Mutex Initialization Scheduling Attributes**

In a priority-driven environment, a direct use of traditional primitives like mutexes and condition variables can lead to unbounded priority inversion, where a higher priority thread can be blocked by a lower priority thread, or set of threads, for an unbounded duration of time. As a result, it becomes impossible to guarantee thread deadlines. Priority inversion can be bounded and minimized by the use of priority inheritance protocols. This allows thread deadlines to be guaranteed even in the presence of synchronization requirements.

Two useful but simple members of the family of priority inheritance protocols are the basic priority inheritance protocol and the priority ceiling protocol emulation. Under the Basic Priority

7216 Inheritance protocol (governed by the Thread Priority Inheritance option), a thread that is  
 7217 blocking higher priority threads executes at the priority of the highest priority thread that it  
 7218 blocks. This simple mechanism allows priority inversion to be bounded by the duration of  
 7219 critical sections and makes timing analysis possible.

7220 Under the Priority Ceiling Protocol Emulation protocol (governed by the Thread Priority  
 7221 Protection option), each mutex has a priority ceiling, usually defined as the priority of the  
 7222 highest priority thread that can lock the mutex. When a thread is executing inside critical  
 7223 sections, its priority is unconditionally increased to the highest of the priority ceilings of all the  
 7224 mutexes owned by the thread. This protocol has two very desirable properties in uni-processor  
 7225 systems. First, a thread can be blocked by a lower priority thread for at most the duration of one  
 7226 single critical section. Furthermore, when the protocol is correctly used in a single processor, and  
 7227 if threads do not become blocked while owning mutexes, mutual deadlocks are prevented.

7228 The priority ceiling emulation can be extended to multiple processor environments, in which  
 7229 case the values of the priority ceilings will be assigned depending on the kind of mutex that is  
 7230 being used: local to only one processor, or global, shared by several processors. Local priority  
 7231 ceilings will be assigned the usual way, equal to the priority of the highest priority thread that  
 7232 may lock that mutex. Global priority ceilings will usually be assigned a priority level higher than  
 7233 all the priorities assigned to any of the threads that reside in the involved processors to avoid the  
 7234 effect called remote blocking.

#### 7235 **Change the Priority Ceiling of a Mutex**

7236 In order for the priority protect protocol to exhibit its desired properties of bounding priority  
 7237 inversion and avoidance of deadlock, it is critical that the ceiling priority of a mutex be the same  
 7238 as the priority of the highest thread that can ever hold it, or higher. Thus, if the priorities of the  
 7239 threads using such mutexes never change dynamically, there is no need ever to change the  
 7240 priority ceiling of a mutex.

7241 However, if a major system mode change results in an altered response time requirement for one  
 7242 or more application threads, their priority has to change to reflect it. It will occasionally be the  
 7243 case that the priority ceilings of mutexes held also need to change. While changing priority  
 7244 ceilings should generally be avoided, it is important that IEEE Std 1003.1-2001 provide these  
 7245 interfaces for those cases in which it is necessary.

#### 7246 **B.2.9.5 Thread Cancellation**

7247 Many existing threads packages have facilities for canceling an operation or canceling a thread.  
 7248 These facilities are used for implementing user requests (such as the CANCEL button in a  
 7249 window-based application), for implementing OR parallelism (for example, telling the other  
 7250 threads to stop working once one thread has found a forced mate in a parallel chess program), or  
 7251 for implementing the ABORT mechanism in Ada.

7252 POSIX programs traditionally have used the signal mechanism combined with either *longjmp()*  
 7253 or polling to cancel operations. Many POSIX programmers have trouble using these facilities to  
 7254 solve their problems efficiently in a single-threaded process. With the introduction of threads,  
 7255 these solutions become even more difficult to use.

7256 The main issues with implementing a cancellation facility are specifying the operation to be  
 7257 canceled, cleanly releasing any resources allocated to that operation, controlling when the target  
 7258 notices that it has been canceled, and defining the interaction between asynchronous signals and  
 7259 cancellation.

## Specifying the Operation to Cancel

Consider a thread that calls through five distinct levels of program abstraction and then, inside the lowest-level abstraction, calls a function that suspends the thread. (An abstraction boundary is a layer at which the client of the abstraction sees only the service being provided and can remain ignorant of the implementation. Abstractions are often layered, each level of abstraction being a client of the lower-level abstraction and implementing a higher-level abstraction.) Depending on the semantics of each abstraction, one could imagine wanting to cancel only the call that causes suspension, only the bottom two levels, or the operation being done by the entire thread. Canceling operations at a finer grain than the entire thread is difficult because threads are active and they may be run in parallel on a multi-processor. By the time one thread can make a request to cancel an operation, the thread performing the operation may have completed that operation and gone on to start another operation whose cancellation is not desired. Thread IDs are not reused until the thread has exited, and either it was created with the *Attr detachstate* attribute set to *PTHREAD\_CREATE\_DETACHED* or the *pthread\_join()* or *pthread\_detach()* function has been called for that thread. Consequently, a thread cancellation will never be misdirected when the thread terminates. For these reasons, the canceling of operations is done at the granularity of the thread. Threads are designed to be inexpensive enough so that a separate thread may be created to perform each separately cancelable operation; for example, each possibly long running user request.

For cancellation to be used in existing code, cancellation scopes and handlers will have to be established for code that needs to release resources upon cancellation, so that it follows the programming discipline described in the text.

## A Special Signal Versus a Special Interface

Two different mechanisms were considered for providing the cancellation interfaces. The first was to provide an interface to direct signals at a thread and then to define a special signal that had the required semantics. The other alternative was to use a special interface that delivered the correct semantics to the target thread.

The solution using signals produced a number of problems. It required the implementation to provide cancellation in terms of signals whereas a perfectly valid (and possibly more efficient) implementation could have both layered on a low-level set of primitives. There were so many exceptions to the special signal (it cannot be used with *kill()*, no POSIX.1 interfaces can be used with it) that it was clearly not a valid signal. Its semantics on delivery were also completely different from any existing POSIX.1 signal. As such, a special interface that did not mandate the implementation and did not confuse the semantics of signals and cancellation was felt to be the better solution.

## Races Between Cancellation and Resuming Execution

Due to the nature of cancellation, there is generally no synchronization between the thread requesting the cancellation of a blocked thread and events that may cause that thread to resume execution. For this reason, and because excess serialization hurts performance, when both an event that a thread is waiting for has occurred and a cancellation request has been made and cancellation is enabled, IEEE Std 1003.1-2001 explicitly allows the implementation to choose between returning from the blocking call or acting on the cancellation request.

## Interaction of Cancellation with Asynchronous Signals

A typical use of cancellation is to acquire a lock on some resource and to establish a cancellation cleanup handler for releasing the resource when and if the thread is canceled.

A correct and complete implementation of cancellation in the presence of asynchronous signals requires considerable care. An implementation has to push a cancellation cleanup handler on the cancellation cleanup stack while maintaining the integrity of the stack data structure. If an asynchronously-generated signal is posted to the thread during a stack operation, the signal handler cannot manipulate the cancellation cleanup stack. As a consequence, asynchronous signal handlers may not cancel threads or otherwise manipulate the cancellation state of a thread. Threads may, of course, be canceled by another thread that used a *sigwait()* function to wait synchronously for an asynchronous signal.

In order for cancellation to function correctly, it is required that asynchronous signal handlers not change the cancellation state. This requires that some elements of existing practice, such as using *longjmp()* to exit from an asynchronous signal handler implicitly, be prohibited in cases where the integrity of the cancellation state of the interrupt thread cannot be ensured.

## Thread Cancellation Overview

### • Cancelability States

The three possible cancelability states (disabled, deferred, and asynchronous) are encoded into two separate bits ((disable, enable) and (deferred, asynchronous)) to allow them to be changed and restored independently. For instance, short code sequences that will not block sometimes disable cancelability on entry and restore the previous state upon exit. Likewise, long or unbounded code sequences containing no convenient explicit cancellation points will sometimes set the cancelability type to asynchronous on entry and restore the previous value upon exit.

### • Cancellation Points

Cancellation points are points inside of certain functions where a thread has to act on any pending cancellation request when cancelability is enabled, if the function would block. As with checking for signals, operations need only check for pending cancellation requests when the operation is about to block indefinitely.

The idea was considered of allowing implementations to define whether blocking calls such as *read()* should be cancellation points. It was decided that it would adversely affect the design of conforming applications if blocking calls were not cancellation points because threads could be left blocked in an uncancelable state.

There are several important blocking routines that are specifically not made cancellation points:

— *pthread\_mutex\_lock()*

If *pthread\_mutex\_lock()* were a cancellation point, every routine that called it would also become a cancellation point (that is, any routine that touched shared state would automatically become a cancellation point). For example, *malloc()*, *free()*, and *rand()* would become cancellation points under this scheme. Having too many cancellation points makes programming very difficult, leading to either much disabling and restoring of cancelability or much difficulty in trying to arrange for reliable cleanup at every possible place.

Since *pthread\_mutex\_lock()* is not a cancellation point, threads could result in being blocked uninterruptibly for long periods of time if mutexes were used as a general

synchronization mechanism. As this is normally not acceptable, mutexes should only be used to protect resources that are held for small fixed lengths of time where not being able to be canceled will not be a problem. Resources that need to be held exclusively for long periods of time should be protected with condition variables.

— `pthread_barrier_wait()`

Canceled a barrier wait will render a barrier unusable. Similar to a barrier timeout (which the standard developers rejected), there is no way to guarantee the consistency of a barrier's internal data structures if a barrier wait is canceled.

— `pthread_spin_lock()`

As with mutexes, spin locks should only be used to protect resources that are held for small fixed lengths of time where not being cancelable will not be a problem.

Every library routine should specify whether or not it includes any cancellation points. Typically, only those routines that may block or compute indefinitely need to include cancellation points.

Correctly coded routines only reach cancellation points after having set up a cancellation cleanup handler to restore invariants if the thread is canceled at that point. Being cancelable only at specified cancellation points allows programmers to keep track of actions needed in a cancellation cleanup handler more easily. A thread should only be made asynchronously cancelable when it is not in the process of acquiring or releasing resources or otherwise in a state from which it would be difficult or impossible to recover.

#### • Thread Cancellation Cleanup Handlers

The cancellation cleanup handlers provide a portable mechanism, easy to implement, for releasing resources and restoring invariants. They are easier to use than signal handlers because they provide a stack of cancellation cleanup handlers rather than a single handler, and because they have an argument that can be used to pass context information to the handler.

The alternative to providing these simple cancellation cleanup handlers (whose only use is for cleaning up when a thread is canceled) is to define a general exception package that could be used for handling and cleaning up after hardware traps and software-detected errors. This was too far removed from the charter of providing threads to handle asynchrony. However, it is an explicit goal of IEEE Std 1003.1-2001 to be compatible with existing exception facilities and languages having exceptions.

The interaction of this facility and other procedure-based or language-level exception facilities is unspecified in this version of IEEE Std 1003.1-2001. However, it is intended that it be possible for an implementation to define the relationship between these cancellation cleanup handlers and Ada, C++, or other language-level exception handling facilities.

It was suggested that the cancellation cleanup handlers should also be called when the process exits or calls the `exec` function. This was rejected partly due to the performance problem caused by having to call the cancellation cleanup handlers of every thread before the operation could continue. The other reason was that the only state expected to be cleaned up by the cancellation cleanup handlers would be the intraprocess state. Any handlers that are to clean up the interprocess state would be registered with `atexit()`. There is the orthogonal problem that the `exec` functions do not honor the `atexit()` handlers, but resolving this is beyond the scope of IEEE Std 1003.1-2001.

- Async-Cancel Safety

A function is said to be async-cancel-safe if it is written in such a way that entering the function with asynchronous cancelability enabled will not cause any invariants to be violated, even if a cancellation request is delivered at any arbitrary instruction. Functions that are async-cancel-safe are often written in such a way that they need to acquire no resources for their operation and the visible variables that they may write are strictly limited.

Any routine that gets a resource as a side effect cannot be made async-cancel-safe (for example, `malloc()`). If such a routine were called with asynchronous cancelability enabled, it might acquire the resource successfully, but as it was returning to the client, it could act on a cancellation request. In such a case, the application would have no way of knowing whether the resource was acquired or not.

Indeed, because many interesting routines cannot be made async-cancel-safe, most library routines in general are not async-cancel-safe. Every library routine should specify whether or not it is async-cancel safe so that programmers know which routines can be called from code that is asynchronously cancelable.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/8 is applied, adding the `pselect()` function to the list of functions with cancellation points.

#### B.2.9.6 Thread Read-Write Locks

##### Background

Read-write locks are often used to allow parallel access to data on multi-processors, to avoid context switches on uni-processors when multiple threads access the same data, and to protect data structures that are frequently accessed (that is, read) but rarely updated (that is, written). The in-core representation of a file system directory is a good example of such a data structure. One would like to achieve as much concurrency as possible when searching directories, but limit concurrent access when adding or deleting files.

Although read-write locks can be implemented with mutexes and condition variables, such implementations are significantly less efficient than is possible. Therefore, this synchronization primitive is included in IEEE Std 1003.1-2001 for the purpose of allowing more efficient implementations in multi-processor systems.

##### Queuing of Waiting Threads

The `pthread_rwlock_unlock()` function description states that one writer or one or more readers must acquire the lock if it is no longer held by any thread as a result of the call. However, the function does not specify which thread(s) acquire the lock, unless the Thread Execution Scheduling option is supported.

The standard developers considered the issue of scheduling with respect to the queuing of threads blocked on a read-write lock. The question turned out to be whether IEEE Std 1003.1-2001 should require priority scheduling of read-write locks for threads whose execution scheduling policy is priority-based (for example, `SCHED_FIFO` or `SCHED_RR`). There are tradeoffs between priority scheduling, the amount of concurrency achievable among readers, and the prevention of writer and/or reader starvation.

For example, suppose one or more readers hold a read-write lock and the following threads request the lock in the listed order:

7433 pthread\_rwlock\_wrlock() - Low priority thread writer\_a  
 7434 pthread\_rwlock\_rdlock() - High priority thread reader\_a  
 7435 pthread\_rwlock\_rdlock() - High priority thread reader\_b  
 7436 pthread\_rwlock\_rdlock() - High priority thread reader\_c

7437 When the lock becomes available, should *writer\_a* block the high priority readers? Or, suppose a  
 7438 read-write lock becomes available and the following are queued:

7439 pthread\_rwlock\_rdlock() - Low priority thread reader\_a  
 7440 pthread\_rwlock\_rdlock() - Low priority thread reader\_b  
 7441 pthread\_rwlock\_rdlock() - Low priority thread reader\_c  
 7442 pthread\_rwlock\_wrlock() - Medium priority thread writer\_a  
 7443 pthread\_rwlock\_rdlock() - High priority thread reader\_d

7444 If priority scheduling is applied then *reader\_d* would acquire the lock and *writer\_a* would block  
 7445 the remaining readers. But should the remaining readers also acquire the lock to increase  
 7446 concurrency? The solution adopted takes into account that when the Thread Execution  
 7447 Scheduling option is supported, high priority threads may in fact starve low priority threads (the  
 7448 application developer is responsible in this case for designing the system in such a way that this  
 7449 starvation is avoided). Therefore, IEEE Std 1003.1-2001 specifies that high priority readers take  
 7450 precedence over lower priority writers. However, to prevent writer starvation from threads of  
 7451 the same or lower priority, writers take precedence over readers of the same or lower priority.

7452 Priority inheritance mechanisms are non-trivial in the context of read-write locks. When a high  
 7453 priority writer is forced to wait for multiple readers, for example, it is not clear which subset of  
 7454 the readers should inherit the writer's priority. Furthermore, the internal data structures that  
 7455 record the inheritance must be accessible to all readers, and this implies some sort of  
 7456 serialization that could negate any gain in parallelism achieved through the use of multiple  
 7457 readers in the first place. Finally, existing practice does not support the use of priority  
 7458 inheritance for read-write locks. Therefore, no specification of priority inheritance or priority  
 7459 ceiling is attempted. If reliable priority-scheduled synchronization is absolutely required, it can  
 7460 always be obtained through the use of mutexes.

## 7461 **Comparison to *fcntl()* Locks**

7462 The read-write locks and the *fcntl()* locks in IEEE Std 1003.1-2001 share a common goal:  
 7463 increasing concurrency among readers, thus increasing throughput and decreasing delay.

7464 However, the read-write locks have two features not present in the *fcntl()* locks. First, under  
 7465 priority scheduling, read-write locks are granted in priority order. Second, also under priority  
 7466 scheduling, writer starvation is prevented by giving writers preference over readers of equal or  
 7467 lower priority.

7468 Also, read-write locks can be used in systems lacking a file system, such as those conforming to  
 7469 the minimal realtime system profile of IEEE Std 1003.13-1998.

## 7470 **History of Resolution Issues**

7471 Based upon some balloting objections, early drafts specified the behavior of threads waiting on a  
 7472 read-write lock during the execution of a signal handler, as if the thread had not called the lock  
 7473 operation. However, this specified behavior would require implementations to establish  
 7474 internal signal handlers even though this situation would be rare, or never happen for many  
 7475 programs. This would introduce an unacceptable performance hit in comparison to the little  
 7476 additional functionality gained. Therefore, the behavior of read-write locks and signals was  
 7477 reverted back to its previous mutex-like specification.



7478 *B.2.9.7 Thread Interactions with Regular File Operations*

7479 There is no additional rationale provided for this section.

7480 **B.2.10 Sockets**

7481 The base document for the sockets interfaces in IEEE Std 1003.1-2001 is the XNS, Issue 5.2  
7482 specification. This was primarily chosen as it aligns with IPv6. Additional material has been  
7483 added from IEEE Std 1003.1g-2000, notably socket concepts, raw sockets, the *pselect()* function,  
7484 the *socketmark()* function, and the `<sys/select.h>` header.

7485 *B.2.10.1 Address Families*

7486 There is no additional rationale provided for this section.

7487 *B.2.10.2 Addressing*

7488 There is no additional rationale provided for this section.

7489 *B.2.10.3 Protocols*

7490 There is no additional rationale provided for this section.

7491 *B.2.10.4 Routing*

7492 There is no additional rationale provided for this section.

7493 *B.2.10.5 Interfaces*

7494 There is no additional rationale provided for this section.

7495 *B.2.10.6 Socket Types*

7496 The type **socklen\_t** was invented to cover the range of implementations seen in the field. The  
7497 intent of **socklen\_t** is to be the type for all lengths that are naturally bounded in size; that is, that  
7498 they are the length of a buffer which cannot sensibly become of massive size: network addresses,  
7499 host names, string representations of these, ancillary data, control messages, and socket options  
7500 are examples. Truly boundless sizes are represented by **size\_t** as in *read()*, *write()*, and so on.

7501 All **socklen\_t** types were originally (in BSD UNIX) of type **int**. During the development of  
7502 IEEE Std 1003.1-2001, it was decided to change all buffer lengths to **size\_t**, which appears at face  
7503 value to make sense. When dual mode 32/64-bit systems came along, this choice unnecessarily  
7504 complicated system interfaces because **size\_t** (with **long**) was a different size under ILP32 and  
7505 LP64 models. Reverting to **int** would have happened except that some implementations had  
7506 already shipped 64-bit-only interfaces. The compromise was a type which could be defined to be  
7507 any size by the implementation: **socklen\_t**.

7508 *B.2.10.7 Socket I/O Mode*

7509 There is no additional rationale provided for this section.

7510 *B.2.10.8 Socket Owner*

7511 There is no additional rationale provided for this section.

7512 *B.2.10.9 Socket Queue Limits*

7513 There is no additional rationale provided for this section.

7514 *B.2.10.10 Pending Error*

7515 There is no additional rationale provided for this section.

7516 *B.2.10.11 Socket Receive Queue*

7517 There is no additional rationale provided for this section.

7518 *B.2.10.12 Socket Out-of-Band Data State*

7519 There is no additional rationale provided for this section.

7520 *B.2.10.13 Connection Indication Queue*

7521 There is no additional rationale provided for this section.

7522 *B.2.10.14 Signals*

7523 There is no additional rationale provided for this section.

7524 *B.2.10.15 Asynchronous Errors*

7525 There is no additional rationale provided for this section.

7526 *B.2.10.16 Use of Options*

7527 There is no additional rationale provided for this section.

7528 *B.2.10.17 Use of Sockets for Local UNIX Connections*

7529 There is no additional rationale provided for this section.

7530 *B.2.10.18 Use of Sockets over Internet Protocols*

7531 A raw socket allows privileged users direct access to a protocol; for example, raw access to the  
7532 IP and ICMP protocols is possible through raw sockets. Raw sockets are intended for  
7533 knowledgeable applications that wish to take advantage of some protocol feature not directly  
7534 accessible through the other sockets interfaces.

7535 *B.2.10.19 Use of Sockets over Internet Protocols Based on IPv4*

7536 There is no additional rationale provided for this section.

7537 *B.2.10.20 Use of Sockets over Internet Protocols Based on IPv6*

7538 The Open Group Base Resolution bwg2001-012 is applied, clarifying that IPv6 implementations  
7539 are required to support use of AF\_INET6 sockets over IPv6.

## 7540 **B.2.11 Tracing**

7541 The organization of the tracing rationale differs from the traditional rationale in that this tracing  
7542 rationale text is written against the trace interface as a whole, rather than against the individual  
7543 components of the trace interface or the normative section in which those components are  
7544 defined. Therefore the sections below do not parallel the sections of normative text in  
7545 IEEE Std 1003.1-2001.

### 7546 *B.2.11.1 Objectives*

7547 The intended uses of tracing are application-system debugging during system development, as a  
7548 “flight recorder” for maintenance of fielded systems, and as a performance measurement tool. In  
7549 all of these intended uses, the vendor-supplied computer system and its software are, for this  
7550 discussion, assumed error-free; the intent being to debug the user-written and/or third-party  
7551 application code, and their interactions. Clearly, problems with the vendor-supplied system and  
7552 its software will be uncovered from time to time, but this is a byproduct of the primary activity,  
7553 debugging user code.

7554 Another need for defining a trace interface in POSIX stems from the objective to provide an  
7555 efficient portable way to perform benchmarks. Existing practice shows that such interfaces are  
7556 commonly used in a variety of systems but with little commonality. As part of the benchmarking  
7557 needs, two aspects within the trace interface must be considered.

7558 The first, and perhaps more important one, is the qualitative aspect.

7559 The second is the quantitative aspect.

#### 7560 • Qualitative Aspect

7561 To better understand this aspect, let us consider an example. Suppose that you want to  
7562 organize a number of actions to be performed during the day. Some of these actions are  
7563 known at the beginning of the day. Some others, which may be more or less important, will  
7564 be triggered by reading your mail. During the day you will make some phone calls and  
7565 synchronously receive some more information. Finally you will receive asynchronous phone  
7566 calls that also will trigger actions. If you, or somebody else, examines your day at work, you,  
7567 or he, can discover that you have not efficiently organized your work. For instance, relative  
7568 to the phone calls you made, would it be preferable to make some of these early in the  
7569 morning? Or to delay some others until the end of the day? Relative to the phone calls you  
7570 have received, you might find that somebody you called in the morning has called you 10  
7571 times while you were performing some important work. To examine, afterwards, your day at  
7572 work, you record in sequence all the trace events relative to your work. This should give you  
7573 a chance of organizing your next day at work.

7574 This is the qualitative aspect of the trace interface. The user of a system needs to keep a trace  
7575 of particular points the application passes through, so that he can eventually make some  
7576 changes in the application and/or system configuration, to give the application a chance of  
7577 running more efficiently.

#### 7578 • Quantitative Aspect

7579 This aspect concerns primarily realtime applications, where missed deadlines can be  
7580 undesirable. Although there are, in IEEE Std 1003.1-2001, some interfaces useful for such  
7581 applications (timeouts, execution time monitoring, and so on), there are no APIs to aid in the  
7582 tuning of a realtime application's behavior (**timespec** in timeouts, length of message queues,  
7583 duration of driver interrupt service routine, and so on). The tuning of an application needs a  
7584 means of recording timestamped important trace events during execution in order to analyze  
7585 offline, and eventually, to tune some realtime features (redesign the system with less

7586 functionalities, readjust timeouts, redesign driver interrupts, and so on).

## 7587 **Detailed Objectives**

7588 Objectives were defined to build the trace interface and are kept for historical interest. Although  
7589 some objectives are not fully respected in this trace interface, the concept of the POSIX trace  
7590 interface assumes the following points:

- 7591 1. It must be possible to trace both system and user trace events concurrently.
- 7592 2. It must be possible to trace per-process trace events and also to trace system trace events  
7593 which are unrelated to any particular process. A per-process trace event is either user-  
7594 initiated or system-initiated.
- 7595 3. It must be possible to control tracing on a per-process basis from either inside or outside  
7596 the process.
- 7597 4. It must be possible to control tracing on a per-thread basis from inside the enclosing  
7598 process.
- 7599 5. Trace points must be controllable by trace event type ID from inside and outside of the  
7600 process. Multiple trace points can have the same trace event type ID, and will be controlled  
7601 jointly.
- 7602 6. Recording of trace events is dependent on both trace event type ID and the  
7603 process/thread. Both must be enabled in order to record trace events. System trace events  
7604 may or may not be handled differently.
- 7605 7. The API must not mandate the ability to control tracing for more than one process at the  
7606 same time.
- 7607 8. There is no objective for trace control on anything bigger than a process; for example,  
7608 group or session.
- 7609 9. Trace propagation and control:
  - 7610 a. Trace propagation across *fork()* is optional; the default is to not trace a child process.
  - 7611 b. Trace control must span *pthread\_create()* operations; that is, if a process is being  
7612 traced, any thread will be traced as well if this thread allows tracing. The default is to  
7613 allow tracing.
- 7614 10. Trace control must not span *exec* or *posix\_spawn()* operations.
- 7615 11. A triggering API is not required. The triggering API is the ability to command or stop  
7616 tracing based on the occurrence of a specific trace event other than a  
7617 POSIX\_TRACE\_START trace event or a POSIX\_TRACE\_STOP trace event.
- 7618 12. Trace log entries must have timestamps of implementation-defined resolution.  
7619 Implementations are exhorted to support at least microsecond resolution. When a trace log  
7620 entry is retrieved, it must have timestamp, PC address, PID, and TID of the entity that  
7621 generated the trace event.
- 7622 13. Independently developed code should be able to use trace facilities without coordination  
7623 and without conflict.
- 7624 14. Even if the trace points in the trace calls are not unique, the trace log entries (after any  
7625 processing) must be uniquely identified as to trace point.
- 7626 15. There must be a standard API to read the trace stream.

- 7627 16. The format of the trace stream and the trace log is opaque and unspecified.
- 7628 17. It must be possible to read a completed trace, if recorded on some suitable non-volatile  
7629 storage, even subsequent to a power cycle or subsequent cold boot of the system.
- 7630 18. Support of analysis of a trace log while it is being formed is implementation-defined.
- 7631 19. The API must allow the application to write trace stream identification information into  
7632 the trace stream and to be able to retrieve it, without it being overwritten by trace entries,  
7633 even if the trace stream is full.
- 7634 20. It must be possible to specify the destination of trace data produced by trace events.
- 7635 21. It must be possible to have different trace streams, and for the tracing enabled by one trace  
7636 stream to be completely independent of the tracing of another trace stream.
- 7637 22. It must be possible to trace events from threads in different CPUs.
- 7638 23. The API must support one or more trace streams per-system, and one or more trace  
7639 streams per-process, up to an implementation-defined set of per-system and per-process  
7640 maximums.
- 7641 24. It must be possible to determine the order in which the trace events happened, without  
7642 necessarily depending on the clock, up to an implementation-defined time resolution.
- 7643 25. For performance reasons, the trace event point call(s) must be implementable as a macro  
7644 (see the ISO POSIX-1: 1996 standard, 1.3.4, Statement 2).
- 7645 26. IEEE Std 1003.1-2001 must not define the trace points which a conforming system must  
7646 implement, except for trace points used in the control of tracing.
- 7647 27. The APIs must be thread-safe, and trace points should be lock-free (that is, not require a  
7648 lock to gain exclusive access to some resource).
- 7649 28. The user-provided information associated with a trace event is variable-sized, up to some  
7650 maximum size.
- 7651 29. Bounds on record and trace stream sizes:
- 7652 a. The API must permit the application to declare the upper bounds on the length of an  
7653 application data record. The system must return the limit it used. The limit used may  
7654 be smaller than requested.
- 7655 b. The API must permit the application to declare the upper bounds on the size of trace  
7656 streams. The system must return the limit it used. The limit used may be different,  
7657 either larger or smaller, than requested.
- 7658 30. The API must be able to pass any fundamental data type, and a structured data type  
7659 composed only of fundamental types. The API must be able to pass data by reference,  
7660 given only as an address and a length. Fundamental types are the POSIX.1 types (see the  
7661 **<sys/types.h>** header) plus those defined in the ISO C standard.
- 7662 31. The API must apply the POSIX notions of ownership and permission to recorded trace  
7663 data, corresponding to the sources of that data.

**Comments on Objectives**

**Note:** In the following comments, numbers in square brackets refer to the above objectives.

It is necessary to be able to obtain a trace stream for a complete activity. Thus there is a requirement to be able to trace both application and system trace events. A per-process trace event is either user-initiated, like the *write()* function, or system-initiated, like a timer expiration. There is also a need to be able to trace an entire process' activity even when it has threads in multiple CPUs. To avoid excess trace activity, it is necessary to be able to control tracing on a trace event type basis.

[Objectives 1,2,5,22]

There is a need to be able to control tracing on a per-process basis, both from inside and outside the process; that is, a process can start a trace activity on itself or any other process. There is also the perceived need to allow the definition of a maximum number of trace streams per system.

[Objectives 3,23]

From within a process, it is necessary to be able to control tracing on a per-thread basis. This provides an additional filtering capability to keep the amount of traced data to a minimum. It also allows for less ambiguity as to the origin of trace events. It is recognized that thread-level control is only valid from within the process itself. It is also desirable to know the maximum number of trace streams per process that can be started. The API should not require thread synchronization or mandate priority inversions that would cause the thread to block. However, the API must be thread-safe.

[Objectives 4,23,24,27]

There was no perceived objective to control tracing on anything larger than a process; for example, a group or session. Also, the ability to start or stop a trace activity on multiple processes atomically may be very difficult or cumbersome in some implementations.

[Objectives 6,8]

It is also necessary to be able to control tracing by trace event type identifier, sometimes called a trace hook ID. However, there is no mandated set of system trace events, since such trace points are implementation-defined. The API must not require from the operating system facilities that are not standard.

[Objectives 6,26]

Trace control must span *fork()* and *pthread\_create()*. If not, there will be no way to ensure that an application's activity is entirely traced. The newly forked child would not be able to turn on its tracing until after it obtained control after the fork, and trace control externally would be even more problematic.

[Objective 9]

Since *exec* and *posix\_spawn()* represent a complete change in the execution of a task (a new program), trace control need not persist over an *exec* or *posix\_spawn()*.

[Objective 10]

Where trace activities are started on multiple processes, these trace activities should not interfere with each other.

[Objective 21]

There is no need for a triggering objective, primarily for performance reasons; see also Section B.2.11.8 (on page 202), rationale on triggering.

[Objective 11]

It must be possible to determine the origin of each traced event. The process and thread identifiers for each trace event are needed. Also there was a perceived need for a user-specifiable origin, but it was felt that this would create too much overhead.

|      |                                                                                                      |
|------|------------------------------------------------------------------------------------------------------|
| 7711 | [Objectives 12,14]                                                                                   |
| 7712 | An allowance must be made for trace points to come embedded in software components from              |
| 7713 | several different sources and vendors without requiring coordination.                                |
| 7714 | [Objective 13]                                                                                       |
| 7715 | There is a requirement to be able to uniquely identify trace points that may have the same trace     |
| 7716 | stream identifier. This is only necessary when a trace report is produced.                           |
| 7717 | [Objectives 12,14]                                                                                   |
| 7718 | Tracing is a very performance-sensitive activity, and will therefore likely be implemented at a      |
| 7719 | low level within the system. Hence the interface must not mandate any particular buffering or        |
| 7720 | storage method. Therefore, a standard API is needed to read a trace stream. Also the interface       |
| 7721 | must not mandate the format of the trace data, and the interface must not assume a trace storage     |
| 7722 | method. Due to the possibility of a monolithic kernel and the possible presence of multiple          |
| 7723 | processes capable of running trace activities, the two kinds of trace events may be stored in two    |
| 7724 | separate streams for performance reasons. A mandatory dump mechanism, common in some                 |
| 7725 | existing practice, has been avoided to allow the implementation of this set of functions on small    |
| 7726 | realtime profiles for which the concept of a file system is not defined. The trace API calls should  |
| 7727 | be implemented as macros.                                                                            |
| 7728 | [Objectives 15,16,25,30]                                                                             |
| 7729 | Since a trace facility is a valuable service tool, the output (or log) of a completed trace stream   |
| 7730 | that is written to permanent storage must be readable on other systems of the type that              |
| 7731 | produced the trace log. Note that there is no objective to be able to interpret a trace log that was |
| 7732 | not successfully completed.                                                                          |
| 7733 | [Objectives 17,18,19]                                                                                |
| 7734 | For trace streams written to permanent storage, a way to specify the destination of the trace        |
| 7735 | stream is needed.                                                                                    |
| 7736 | [Objective 20]                                                                                       |
| 7737 | There is a requirement to be able to depend on the ordering of trace events up to some               |
| 7738 | implementation-defined time interval. For example, there is a need to know the time period           |
| 7739 | during which, if trace events are closer together, their ordering is unspecified. Events that occur  |
| 7740 | within an interval smaller than this resolution may or may not be read back in the correct order.    |
| 7741 | [Objective 24]                                                                                       |
| 7742 | The application should be able to know how much data can be traced. When trace event types           |
| 7743 | can be filtered, the application should be able to specify the approximate maximum amount of         |
| 7744 | data that will be traced in a trace event so resources can be more efficiently allocated.            |
| 7745 | [Objectives 28,29]                                                                                   |
| 7746 | Users should not be able to trace data to which they would not normally have access. System          |
| 7747 | trace events corresponding to a process/thread should be associated with the ownership of that       |
| 7748 | process/thread.                                                                                      |
| 7749 | [Objective 31]                                                                                       |

## 7750 B.2.11.2 Trace Model

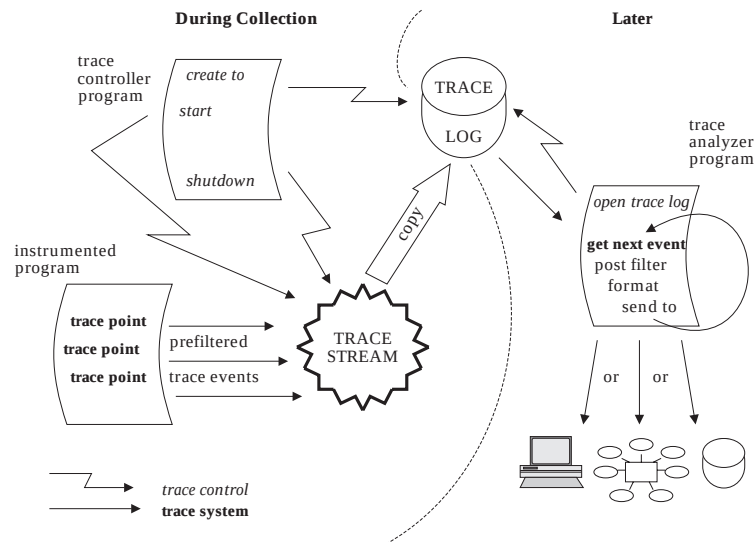
7751 **Introduction**

7752 The model is based on two base entities: the “Trace Stream” and the “Trace Log”, and a recorded  
 7753 unit called the “Trace Event”. The possibility of using Trace Streams and Trace Logs separately  
 7754 gives two use dimensions and solves both the performance issue and the full-information  
 7755 system issue. In the case of a trace stream without log, specific information, although reduced in  
 7756 quantity, is required to be registered, in a possibly small realtime system, with as little overhead  
 7757 as possible. The Trace Log option has been added for small realtime systems. In the case of a  
 7758 trace stream with log, considerable complex application-specific information needs to be  
 7759 collected.

7760 **Trace Model Description**

7761 The trace model can be examined for three different subfunctions: Application Instrumentation,  
 7762 Trace Operation Control, and Trace Analysis.

7763



7764

**Figure B-2** Trace System Overview: for Offline Analysis

7765 Each of these subfunctions requires specific characteristics of the trace mechanism API.

## 7766 • Application Instrumentation

7767 When instrumenting an application, the programmer is not concerned about the future use of  
 7768 the trace events in the trace stream or the trace log, the full policy of the trace stream, or the  
 7769 eventual pre-filtering of trace events. But he is concerned about the correct determination of  
 7770 the specific trace event type identifier, regardless of how many independent libraries are  
 7771 used in the same user application; see Figure B-2 and Figure B-3 (on page 185).

7772 This trace API provides the necessary operations to accomplish this subfunction. This is done  
 7773 by providing functions to associate a programmer-defined name with an implementation-  
 7774 defined trace event type identifier (see the `posix_trace_eventid_open()` function), and to send  
 7775 this trace event into a potential trace stream (see the `posix_trace_event()` function).



• Trace Operation Control

When controlling the recording of trace events in a trace stream, the programmer is concerned with the correct initialization of the trace mechanism (that is, the sizing of the trace stream), the correct retention of trace events in a permanent storage, the correct dynamic recording of trace events, and so on.

This trace API provides the necessary material to permit this efficiently. This is done by providing functions to initialize a new trace stream, and optionally a trace log:

- Trace Stream Attributes Object Initialization (see *posix\_trace\_attr\_init()*)
- Functions to Retrieve or Set Information About a Trace Stream (see *posix\_trace\_attr\_getgenversion()*)
- Functions to Retrieve or Set the Behavior of a Trace Stream (see *posix\_trace\_attr\_getinherited()*)
- Functions to Retrieve or Set Trace Stream Size Attributes (see *posix\_trace\_attr\_getmaxusereventsize()*)
- Trace Stream Initialization, Flush, and Shutdown from a Process (see *posix\_trace\_create()*)
- Clear Trace Stream and Trace Log (see *posix\_trace\_clear()*)

To select the trace event types that are to be traced:

- Manipulate Trace Event Type Identifier (see *posix\_trace\_trid\_eventid\_open()*)
- Iterate over a Mapping of Trace Event Type (see *posix\_trace\_eventtypelist\_getnext\_id()*)
- Manipulate Trace Event Type Sets (see *posix\_trace\_eventset\_empty()*)
- Set Filter of an Initialized Trace Stream (see *posix\_trace\_set\_filter()*)

To control the execution of an active trace stream:

- Trace Start and Stop (see *posix\_trace\_start()*)
- Functions to Retrieve the Trace Attributes or Trace Statuses (see *posix\_trace\_get\_attr()*)

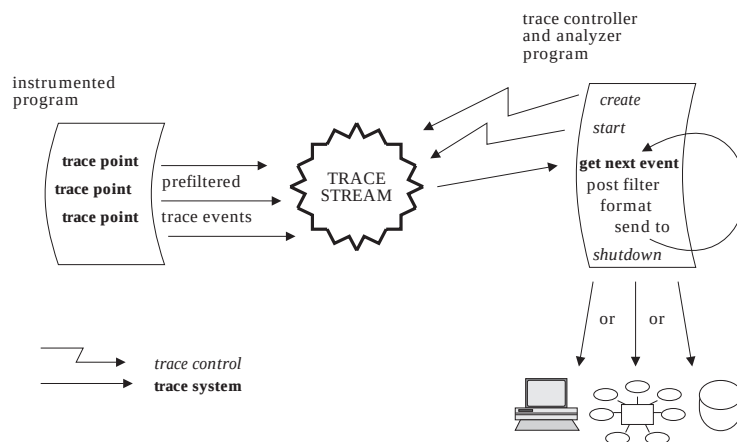


**Figure B-3** Trace System Overview: for Online Analysis

• Trace Analysis

Once correctly recorded, on permanent storage or not, an ultimate activity consists of the analysis of the recorded information. If the recorded data is on permanent storage, a specific open operation is required to associate a trace stream to a trace log.

The first intent of the group was to request the presence of a system identification structure in the trace stream attribute. This was, for the application, to allow some portable way to process the recorded information. However, there is no requirement that the **utsname** structure, on which this system identification was based, be portable from one machine to another, so the contents of the attribute cannot be interpreted correctly by an application conforming to IEEE Std 1003.1-2001.

This modification has been incorporated and requests that some unspecified information be recorded in the trace log in order to fail opening it if the analysis process and the controller process were running in different types of machine, but does not request that this information be accessible to the application. This modification has implied a modification in the *posix\_trace\_open()* function error code returns.

This trace API provides functions to:

- Extract trace stream identification attributes (see *posix\_trace\_attr\_getgenversion()*)
- Extract trace stream behavior attributes (see *posix\_trace\_attr\_getinherited()*)
- Extract trace event, stream, and log size attributes (see *posix\_trace\_attr\_getmaxusereventsize()*)
- Look up trace event type names (see *posix\_trace\_eventid\_get\_name()*)
- Iterate over trace event type identifiers (see *posix\_trace\_eventtypelist\_getnext\_id()*)
- Open, rewind, and close a trace log (see *posix\_trace\_open()*)
- Read trace stream attributes and status (see *posix\_trace\_get\_attr()*)
- Read trace events (see *posix\_trace\_getnext\_event()*)

Due to the following two reasons:

1. The requirement that the trace system must not add unacceptable overhead to the traced process and so that the trace event point execution must be fast
2. The traced application does not care about tracing errors

the trace system cannot return any internal error to the application. Internal error conditions can range from unrecoverable errors that will force the active trace stream to abort, to small errors that can affect the quality of tracing without aborting the trace stream. The group decided to define a system trace event to report to the analysis process such internal errors. It is not the intention of IEEE Std 1003.1-2001 to require an implementation to report an internal error that corrupts or terminates tracing operation. The implementor is free to decide which internal documented errors, if any, the trace system is able to report.

### States of a Trace Stream

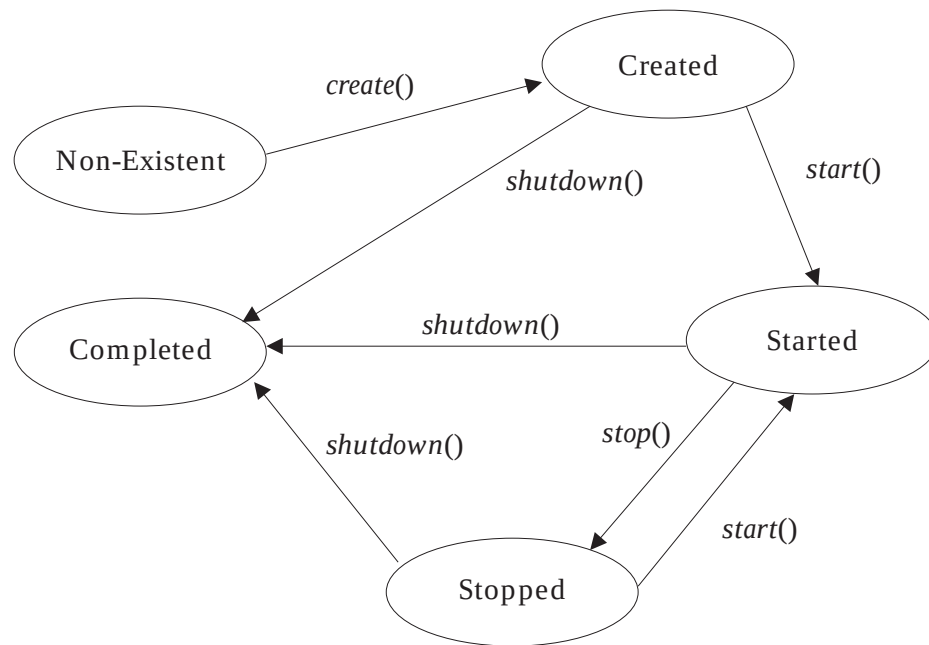


**Figure B-4** Trace System Overview: States of a Trace Stream

Figure B-4 shows the different states an active trace stream passes through. After the `posix_trace_create()` function call, a trace stream becomes `CREATED` and a trace stream is associated for the future collection of trace events. The status of the trace stream is `POSIX_TRACE_SUSPENDED`. The state becomes `STARTED` after a call to the `posix_trace_start()` function, and the status becomes `POSIX_TRACE_RUNNING`. In this state, all trace events that are not filtered out will be stored into the trace stream. After a call to `posix_trace_stop()`, the trace stream becomes `STOPPED` (and the status `POSIX_TRACE_SUSPENDED`). In this state, no new trace events will be recorded in the trace stream, but previously recorded trace events may continue to be read.

After a call to `posix_trace_shutdown()`, the trace stream is in the state `COMPLETED`. The trace stream no longer exists but, if the Trace Log option is supported, all the information contained in it has been logged. If a log object has not been associated with the trace stream at the creation, it is the responsibility of the trace controller process to not shut the trace stream down while trace events remain to be read in the stream.

### Tracing All Processes

Some implementations have a tracing subsystem with the ability to trace all processes. This is useful to debug some types of device drivers such as those for ATM or X25 adapters. These types of adapters are used by several independent processes, that are not issued from the same process.

The POSIX trace interface does not define any constant or option to create a trace stream tracing all processes. POSIX.1 does not prevent this type of implementation and an implementor is free to add this capability. Nevertheless, the trace interface allows tracing of all the system trace events and all the processes issued from the same process.

If such a tracing system capability has to be implemented, when a trace stream is created, it is recommended that a constant named `POSIX_TRACE_ALLPROC` be used instead of the process identifier in the argument of the `posix_trace_create()` or `posix_trace_create_withlog()` function. A possible value for `POSIX_TRACE_ALLPROC` may be `-1` instead of a real process identifier.

The implementor has to be aware that there is some impact on the tracing behavior as defined in the POSIX trace interface. For example:

- If the default value for the inheritance attribute is set to `POSIX_TRACE_CLOSE_FOR_CHILD`, the implementation has to stop tracing for the child process.
- The trace controller which is creating this type of trace stream must have the appropriate privilege to trace all the processes.

### Trace Storage

The model is based on two types of trace events: system trace events and user-defined trace events. The internal representation of trace events is implementation-defined, and so the implementor is free to choose the more suitable, practical, and efficient way to design the internal management of trace events. For the timestamping operation, the model does not impose the `CLOCK_REALTIME` or any other clock. The buffering allocation and operation follow the same principle. The implementor is free to use one or more buffers to record trace events; the interface assumes only a logical trace stream of sequentially recorded trace events. Regarding flushing of trace events, the interface allows the definition of a trace log object which typically can be a file. But the group was also aware of defining functions to permit the use of this interface in small realtime systems, which may not have general file system capabilities. For instance, the three functions `posix_trace_getnext_event()` (blocking), `posix_trace_timedgetnext_event()` (blocking with timeout), and `posix_trace_trygetnext_event()` (non-blocking) are proposed to read the recorded trace events.

The policy to be used when the trace stream becomes full also relies on common practice:

- For an active trace stream, the `POSIX_TRACE_LOOP` trace stream policy permits automatic overrun (overwrite of oldest trace events) while waiting for some user-defined condition to cause tracing to stop. By contrast, the `POSIX_TRACE_UNTIL_FULL` trace stream policy requires the system to stop tracing when the trace stream is full. However, if the trace stream that is full is at least partially emptied by a call to the `posix_trace_flush()` function or by calls to the `posix_trace_getnext_event()` function, the trace system will automatically resume tracing.

If the Trace Log option is supported, the operation of the `POSIX_TRACE_FLUSH` policy is an extension of the `POSIX_TRACE_UNTIL_FULL` policy. The automatic free operation (by flushing to the associated trace log) is added.

- If a log is associated with the trace stream and this log is a regular file, these policies also apply for the log. One more policy, `POSIX_TRACE_APPEND`, is defined to allow indefinite extension of the log. Since the log destination can be any device or pseudo-device, the implementation may not be able to manipulate the destination as required by IEEE Std 1003.1-2001. For this reason, the behavior of the log full policy may be unspecified depending on the trace log type.

The current trace interface does not define a service to preallocate space for a trace log file, because this space can be preallocated by means of a call to the `posix_fallocate()` function. This function could be called after the file has been opened, but before the trace stream is created. The `posix_fallocate()` function ensures that any required storage for regular file data is allocated on the file system storage media. If `posix_fallocate()` returns successfully,

7911 subsequent writes to the specified file data will not fail due to the lack of free space on the file  
 7912 system storage media. Besides trace events, a trace stream also includes trace attributes and  
 7913 the mapping from trace event names to trace event type identifiers. The implementor is free  
 7914 to choose how to store the trace attributes and the trace event type map, but must ensure that  
 7915 this information is not lost when a trace stream overrun occurs.

### 7916 *B.2.11.3 Trace Programming Examples*

7917 Several programming examples are presented to show the code of the different possible  
 7918 subfunctions using a trace subsystem. All these programs need to include the **<trace.h>** header.  
 7919 In the examples shown, error checking is omitted for more simplicity.

### 7920 **Trace Operation Control**

7921 These examples show the creation of a trace stream for another process; one which is already  
 7922 trace instrumented. All the default trace stream attributes are used to simplify programming in  
 7923 the first example. The second example shows more possibilities.

### 7924 **First Example**

```

7925 /* Caution. Error checks omitted */
7926 {
7927 trace_attr_t attr;
7928 pid_t pid = traced_process_pid;
7929 int fd;
7930 trace_id_t trid;
7931
7932 - - - - -
7933 /* Initialize trace stream attributes */
7934 posix_trace_attr_init(&attr);
7935 /* Open a trace log */
7936 fd=open("/tmp/mytracelog",...);
7937 /*
7938 * Create a new trace associated with a log
7939 * and with default attributes
7940 */
7941 posix_trace_create_withlog(pid, &attr, fd, &trid);
7942
7943 /* Trace attribute structure can now be destroyed */
7944 posix_trace_attr_destroy(&attr);
7945 /* Start of trace event recording */
7946 posix_trace_start(trid);
7947 - - - - -
7948 /* Duration of tracing */
7949 - - - - -
7950 /* Stop and shutdown of trace activity */
7951 posix_trace_shutdown(trid);
7952 - - - - -
7953 }
```

**Second Example**

Between the initialization of the trace stream attributes and the creation of the trace stream, these trace stream attributes may be modified; see **Trace Stream Attribute Manipulation** (on page 194) for a specific programming example. Between the creation and the start of the trace stream, the event filter may be set; after the trace stream is started, the event filter may be changed. The setting of an event set and the change of a filter is shown in **Create a Trace Event Type Set and Change the Trace Event Type Filter** (on page 194).

```

/* Caution. Error checks omitted */
{
 trace_attr_t attr;
 pid_t pid = traced_process_pid;
 int fd;
 trace_id_t trid;
 - - - - -
 /* Initialize trace stream attributes */
 posix_trace_attr_init(&attr);
 /* Attr default may be changed at this place; see example */
 - - - - -
 /* Create and open a trace log with R/W user access */
 fd=open("/tmp/mytracelog",O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR);
 /* Create a new trace associated with a log */
 posix_trace_create_withlog(pid, &attr, fd, &trid);
 /*
 * If the Trace Filter option is supported
 * trace event type filter default may be changed at this place;
 * see example about changing the trace event type filter
 */
 posix_trace_start(trid);
 - - - - -

 /*
 * If you have an uninteresting part of the application
 * you can stop temporarily.
 *
 * posix_trace_stop(trid);
 * - - - - -
 * - - - - -
 * posix_trace_start(trid);
 */
 - - - - -

 /*
 * If the Trace Filter option is supported
 * the current trace event type filter can be changed
 * at any time (see example about how to set
 * a trace event type filter)
 */
 - - - - -

 /* Stop the recording of trace events */
 posix_trace_stop(trid);
 /* Shutdown the trace stream */
 posix_trace_shutdown(trid);

```

```

8004 /*
8005 * Destroy trace stream attributes; attr structure may have
8006 * been used during tracing to fetch the attributes
8007 */
8008 posix_trace_attr_destroy(&attr);
8009 - - - - -
8010 }

```

### 8011 **Application Instrumentation**

8012 This example shows an instrumented application. The code is included in a block of instructions,  
 8013 perhaps a function from a library. Possibly in an initialization part of the instrumented  
 8014 application, two user trace events names are mapped to two trace event type identifiers  
 8015 (function *posix\_trace\_eventid\_open()*). Then two trace points are programmed.

```

8016 /* Caution. Error checks omitted */
8017 {
8018 trace_event_id_t eventid1, eventid2;
8019 - - - - -
8020 /* Initialization of two trace event type ids */
8021 posix_trace_eventid_open("my_first_event",&eventid1);
8022 posix_trace_eventid_open("my_second_event",&eventid2);
8023 - - - - -
8024 - - - - -
8025 - - - - -
8026 /* Trace point */
8027 posix_trace_event(eventid1,NULL,0);
8028 - - - - -
8029 /* Trace point */
8030 posix_trace_event(eventid2,NULL,0);
8031 - - - - -
8032 }

```

### 8033 **Trace Analyzer**

8034 This example shows the manipulation of a trace log resulting from the dumping of a completed  
 8035 trace stream. All the default attributes are used to simplify programming, and data associated  
 8036 with a trace event is not shown in the first example. The second example shows more  
 8037 possibilities.

### 8038 **First Example**

```

8039 /* Caution. Error checks omitted */
8040 {
8041 int fd;
8042 trace_id_t trid;
8043 posix_trace_event_info trace_event;
8044 char trace_event_name[TRACE_EVENT_NAME_MAX];
8045 int return_value;
8046 size_t returndatasize;
8047 int lost_event_number;
8048 - - - - -

```

```

8049 /* Open an existing trace log */
8050 fd=open("/tmp/tracelog", O_RDONLY);
8051 /* Open a trace stream on the open log */
8052 posix_trace_open(fd, &trid);
8053 /* Read a trace event */
8054 posix_trace_getnext_event(trid, &trace_event,
8055 NULL, 0, &returndatasize,&return_value);

8056 /* Read and print all trace event names out in a loop */
8057 while (return_value == NULL)
8058 {
8059 /*
8060 * Get the name of the trace event associated
8061 * with trid trace ID
8062 */
8063 posix_trace_eventid_get_name(trid, trace_event.event_id,
8064 trace_event_name);
8065 /* Print the trace event name out */
8066 printf("%s\n",trace_event_name);
8067 /* Read a trace event */
8068 posix_trace_getnext_event(trid, &trace_event,
8069 NULL, 0, &returndatasize,&return_value);
8070 }

8071 /* Close the trace stream */
8072 posix_trace_close(trid);
8073 /* Close the trace log */
8074 close(fd);
8075 }

```

## 8076 Second Example

8077 The complete example includes the two other examples in **Retrieve Information from a Trace**  
8078 **Log** (on page 195) and in **Retrieve the List of Trace Event Types Used in a Trace Log** (on page  
8079 196). For example, the *maxdatasize* variable is set in **Retrieve the List of Trace Event Types Used**  
8080 **in a Trace Log** (on page 196).

```

8081 /* Caution. Error checks omitted */
8082 {
8083 int fd;
8084 trace_id_t trid;
8085 posix_trace_event_info trace_event;
8086 char trace_event_name[TRACE_EVENT_NAME_MAX];
8087 char * data;
8088 size_t maxdatasize=1024, returndatasize;
8089 int return_value;
8090 - - - - -

8091 /* Open an existing trace log */
8092 fd=open("/tmp/tracelog", O_RDONLY);
8093 /* Open a trace stream on the open log */
8094 posix_trace_open(fd, &trid);
8095 /*
8096 * Retrieve information about the trace stream which

```



```

8097 * was dumped in this trace log (see example)
8098 */
8099 - - - - -
8100 /* Allocate a buffer for trace event data */
8101 data=(char *)malloc(maxdatasize);
8102 /*
8103 * Retrieve the list of trace events used in this
8104 * trace log (see example)
8105 */
8106 - - - - -
8107 /* Read and print all trace event names and data out in a loop */
8108 while (1)
8109 {
8110 posix_trace_getnext_event(trid, &trace_event,
8111 data, maxdatasize, &returndatasize,&return_value);
8112 if (return_value != NULL) break;
8113 /*
8114 * Get the name of the trace event type associated
8115 * with trid trace ID
8116 */
8117 posix_trace_eventid_get_name(trid, trace_event.event_id,
8118 trace_event_name);
8119 {
8120 int i;
8121 /* Print the trace event name out */
8122 printf("%s: ", trace_event_name);
8123 /* Print the trace event data out */
8124 for (i=0; i<returndatasize, i++) printf("%02.2X",
8125 (unsigned char)data[i]);
8126 printf("\n");
8127 }
8128 }
8129 /* Close the trace stream */
8130 posix_trace_close(trid);
8131 /* The buffer data is deallocated */
8132 free(data);
8133 /* Now the file can be closed */
8134 close(fd);
8135 }

```

### Several Programming Manipulations

The following examples show some typical sets of operations needed in some contexts.

**Trace Stream Attribute Manipulation**

This example shows the manipulation of a trace stream attribute object in order to change the default value provided by a previous *posix\_trace\_attr\_init()* call.

```

/* Caution. Error checks omitted */
{
 trace_attr_t attr;
 size_t logsize=100000;
 - - - - -
 /* Initialize trace stream attributes */
 posix_trace_attr_init(&attr);
 /* Set the trace name in the attributes structure */
 posix_trace_attr_setname(&attr, "my_trace");
 /* Set the trace full policy */
 posix_trace_attr_setstreamfullpolicy(&attr, POSIX_TRACE_LOOP);
 /* Set the trace log size */
 posix_trace_attr_setlogsize(&attr, logsize);
 - - - - -
}

```

**Create a Trace Event Type Set and Change the Trace Event Type Filter**

This example is valid only if the Trace Event Filter option is supported. This example shows the manipulation of a trace event type set in order to change the trace event type filter for an existing active trace stream, which may be just-created, running, or suspended. Some sets of trace event types are well-known, such as the set of trace event types not associated with a process, some trace event types are just-built trace event types for this trace stream; one trace event type is the predefined trace event error type which is deleted from the trace event type set.

```

/* Caution. Error checks omitted */
{
 trace_id_t trid = existing_trace;
 trace_event_set_t set;
 trace_event_id_t trace_event1, trace_event2;
 - - - - -
 /* Initialize to an empty set of trace event types */
 /* (not strictly required because posix_trace_event_set_fill() */
 /* will ignore the prior contents of the event set.) */
 posix_trace_eventset_emptyset(&set);
 /*
 * Fill the set with all system trace events
 * not associated with a process
 */
 posix_trace_eventset_fill(&set, POSIX_TRACE_WOPID_EVENTS);

 /*
 * Get the trace event type identifier of the known trace event name
 * my_first_event for the trid trace stream
 */
 posix_trace_trid_eventid_open(trid, "my_first_event", &trace_event1);
 /* Add the set with this trace event type identifier */
 posix_trace_eventset_add_event(trace_event1, &set);
 /*

```

```

8186 * Get the trace event type identifier of the known trace event name
8187 * my_second_event for the trid trace stream
8188 */
8189 posix_trace_trid_eventid_open(trid, "my_second_event", &trace_event2);
8190 /* Add the set with this trace event type identifier */
8191 posix_trace_eventset_add_event(trace_event2, &set);
8192 - - - - -
8193 /* Delete the system trace event POSIX_TRACE_ERROR from the set */
8194 posix_trace_eventset_del_event(POSIX_TRACE_ERROR, &set);
8195 - - - - -
8196
8197 /* Modify the trace stream filter making it equal to the new set */
8198 posix_trace_set_filter(trid, &set, POSIX_TRACE_SET_EVENTSET);
8199 - - - - -
8200 /*
8201 * Now trace_event1, trace_event2, and all system trace event types
8202 * not associated with a process, except for the POSIX_TRACE_ERROR
8203 * system trace event type, are filtered out of (not recorded in) the
8204 * existing trace stream.
8205 */
8206 }

```

## Retrieve Information from a Trace Log

This example shows how to extract information from a trace log, the dump of a trace stream. This code:

- Asks if the trace stream has lost trace events
- Extracts the information about the version of the trace subsystem which generated this trace log
- Retrieves the maximum size of trace event data; this may be used to dynamically allocate an array for extracting trace event data from the trace log without overflow

```

8209 /* Caution. Error checks omitted */
8210 {
8211 struct posix_trace_status_info statusinfo;
8212 trace_attr_t attr;
8213 trace_id_t trid = existing_trace;
8214 size_t maxdatasize;
8215 char genversion[TRACE_NAME_MAX];
8216 - - - - -
8217 /* Get the trace stream status */
8218 posix_trace_get_status(trid, &statusinfo);
8219 /* Detect an overrun condition */
8220 if (statusinfo.posix_stream_overrun_status == POSIX_TRACE_OVERRUN)
8221 printf("trace events have been lost\n");
8222
8223 /* Get attributes from the trid trace stream */
8224 posix_trace_get_attr(trid, &attr);
8225 /* Get the trace generation version from the attributes */
8226 posix_trace_attr_getgenversion(&attr, genversion);
8227 /* Print the trace generation version out */
8228 printf("Information about Trace Generator:%s\n", genversion);
8229 }

```

```

8233 /* Get the trace event max data size from the attributes */
8234 posix_trace_attr_getmaxdatasize(&attr, &maxdatasize);
8235 /* Print the trace event max data size out */
8236 printf("Maximum size of associated data:%d\n",maxdatasize);
8237 /* Destroy the trace stream attributes */
8238 posix_trace_attr_destroy(&attr);
8239 }

```

#### 8240 **Retrieve the List of Trace Event Types Used in a Trace Log**

8241 This example shows the retrieval of a trace stream's trace event type list. This operation may be  
 8242 very useful if you are interested only in tracking the type of trace events in a trace log.

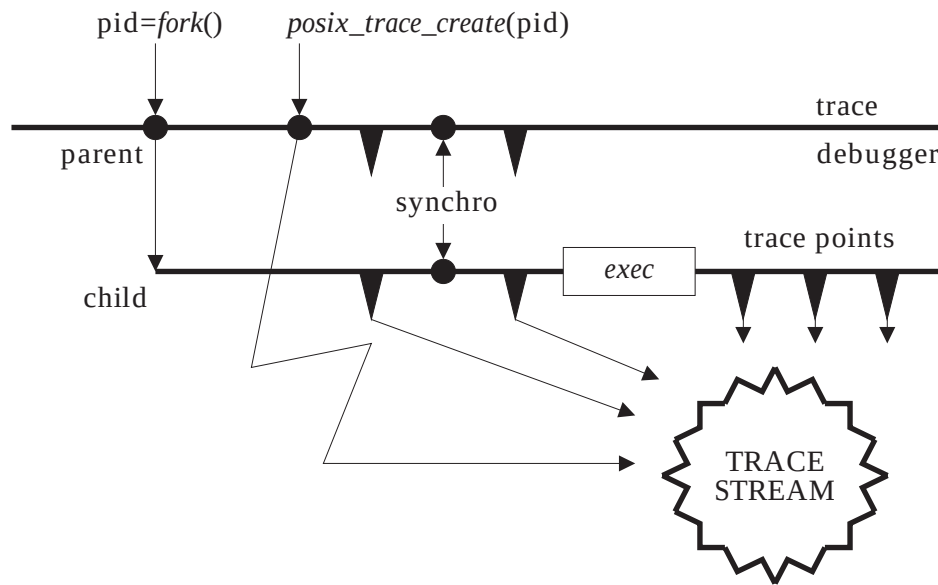
```

8243 /* Caution. Error checks omitted */
8244 {
8245 trace_id_t trid = existing_trace;
8246 trace_event_id_t event_id;
8247 char event_name[TRACE_EVENT_NAME_MAX];
8248 int return_value;
8249 - - - - -
8250
8251 /*
8252 * In a loop print all existing trace event names out
8253 * for the trid trace stream
8254 */
8255 while (1)
8256 {
8257 posix_trace_eventtypelist_getnext_id(trid, &event_id
8258 &return_value);
8259 if (return_value != NULL) break;
8260 /*
8261 * Get the name of the trace event associated
8262 * with trid trace ID
8263 */
8264 posix_trace_eventid_get_name(trid, event_id, event_name);
8265 /* Print the name out */
8266 printf("%s\n", event_name);
8267 }
8268 }

```

## 8268 B.2.11.4 Rationale on Trace for Debugging

8269



8270

**Figure B-5** Trace Another Process

8271

8272

8273

8274

8275

8276

8277

Among the different possibilities offered by the trace interface defined in IEEE Std 1003.1-2001, the debugging of an application is the most interesting one. Typical operations in the controlling debugger process are to filter trace event types, to get trace events from the trace stream, to stop the trace stream when the debugged process is executing uninteresting code, to start the trace stream when some interesting point is reached, and so on. The interface defined in IEEE Std 1003.1-2001 should define all the necessary base functions to allow this dynamic debug handling.

8278

8279

8280

8281

Figure B-5 shows an example in which the trace stream is created after the call to the `fork()` function. If the user does not want to lose trace events, some synchronization mechanism (represented in the figure) may be needed before calling the `exec` function, to give the parent a chance to create the trace stream before the child begins the execution of its trace points.

## 8282 B.2.11.5 Rationale on Trace Event Type Name Space

8283

8284

8285

8286

8287

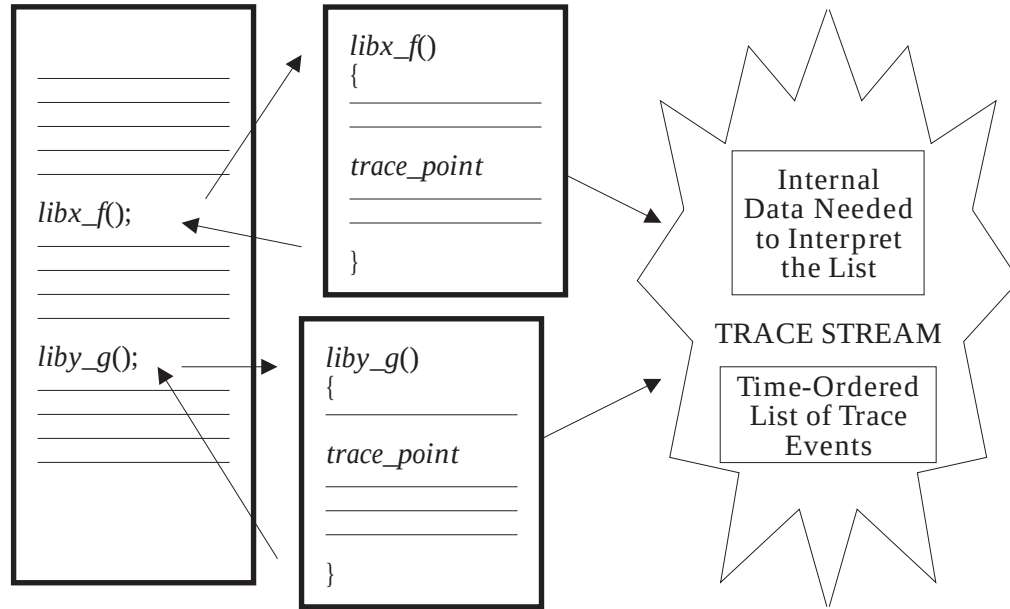
8288

8289

8290

At first, the working group was in favor of the representation of a trace event type by an integer (`event_name`). It seems that existing practice shows the weakness of such a representation. The collision of trace event types is the main problem that cannot be simply resolved using this sort of representation. Suppose, for example, that a third party designs an instrumented library. The user does not have the source of this library and wants to trace his application which uses in some part the third-party library. There is no means for him to know what are the trace event types used in the instrumented library so he has some chance of duplicating some of them and thus to obtain a contaminated tracing of his application.

8291



8292

**Figure B-6** Trace Name Space Overview: With Third-Party Library

8293

8294

8295

8296

8297

8298

8299

8300

8301

8302

8303

There are requirements to allow program images containing pieces from various vendors to be traced without also requiring those of any other vendors to coordinate their uses of the trace facility, and especially the naming of their various trace event types and trace point IDs. The chosen solution is to provide a very large name space, large enough so that the individual vendors can give their trace types and tracepoint IDs sufficiently long and descriptive names making the occurrence of collisions quite unlikely. The probability of collision is thus made sufficiently low so that the problem may, as a practical matter, be ignored. By requirement, the consequence of collisions will be a slight ambiguity in the trace streams; tracing will continue in spite of collisions and ambiguities. “The show must go on”. The *posix\_prog\_address* member of the **posix\_trace\_event\_info** structure is used to allow trace streams to be unambiguously interpreted, despite the fact that trace event types and trace event names need not be unique.

8304

8305

8306

8307

8308

8309

8310

8311

8312

8313

The *posix\_trace\_eventid\_open()* function is required to allow the instrumented third-party library to get a valid trace event type identifier for its trace event names. This operation is, somehow, an allocation, and the group was aware of proposing some deallocation mechanism which the instrumented application could use to recover the resources used by a trace event type identifier. This would have given the instrumented application the benefit of being capable of reusing a possible minimum set of trace event type identifiers, but also the inconvenience to have, possibly in the same trace stream, one trace event type identifier identifying two different trace event types. After some discussions the group decided to not define such a function which would make this API thicker for little benefit, the user having always the possibility of adding identification information in the *data* member of the trace event structure.

8314

8315

8316

8317

8318

8319

8320

The set of the trace event type identifiers the controlling process wants to filter out is initialized in the trace mechanism using the function *posix\_trace\_set\_filter()*, setting the arguments according to the definitions explained in *posix\_trace\_set\_filter()*. This operation can be done statically (when the trace is in the STOPPED state) or dynamically (when the trace is in the STARTED state). The preparation of the filter is normally done using the function defined in *posix\_trace\_eventtypelist\_getnext\_id()* and eventually the function *posix\_trace\_eventtypelist\_rewind()* in order to know (before the recording) the list of the potential

set of trace event types that can be recorded. In the case of an active trace stream, this list may not be exhaustive. Actually, the target process may not have yet called the function `posix_trace_eventid_open()`. But it is a common practice, for a controlling process, to prepare the filtering of a future trace stream before its start. Therefore the user must have a way to get the trace event type identifier corresponding to a well-known trace event name before its future association by the pre-cited function. This is done by calling the `posix_trace_trid_eventid_open()` function, given the trace stream identifier and the trace name, and described hereafter. Because this trace event type identifier is associated with a trace stream identifier, where a unique process has initialized two or more traces, the implementation is expected to return the same trace event type identifier for successive calls to `posix_trace_trid_eventid_open()` with different trace stream identifiers. The `posix_trace_eventid_get_name()` function is used by the controller process to identify, by the name, the trace event type returned by a call to the `posix_trace_eventtypelist_getnext_id()` function.

Afterwards, the set of trace event types is constructed using the functions defined in `posix_trace_eventset_empty()`, `posix_trace_eventset_fill()`, `posix_trace_eventset_add()`, and `posix_trace_eventset_del()`.

A set of functions is provided devoted to the manipulation of the trace event type identifier and names for an active trace stream. All these functions require the trace stream identifier argument as the first parameter. The opacity of the trace event type identifier implies that the user cannot associate directly its well-known trace event name with the system-associated trace event type identifier.

The `posix_trace_trid_eventid_open()` function allows the application to get the system trace event type identifier back from the system, given its well-known trace event name. This function is useful only when a controlling process needs to specify specific events to be filtered.

The `posix_trace_eventid_get_name()` function allows the application to obtain a trace event name given its trace event type identifier. One possible use of this function is to identify the type of a trace event retrieved from the trace stream, and print it. The easiest way to implement this requirement, is to use a single trace event type map for all the processes whose maps are required to be identical. A more difficult way is to attempt to keep multiple maps identical at every call to `posix_trace_eventid_open()` and `posix_trace_trid_eventid_open()`.

#### B.2.11.6 Rationale on Trace Events Type Filtering

The most basic rationale for runtime and pre-registration filtering (selection/rejection) of trace event types is to prevent choking of the trace collection facility, and/or overloading of the computer system. Any worthwhile trace facility can bring even the largest computer to its knees. Otherwise, everything would be recorded and filtered after the fact; it would be much simpler, but impractical.

To achieve debugging, measurement, or whatever the purpose of tracing, the filtering of trace event types is an important part of trace analysis. Due to the fact that the trace events are put into a trace stream and probably logged afterwards into a file, different levels of filtering—that is, rejection of trace event types—are possible.

**Filtering of Trace Event Types Before Tracing**

This function, represented by the `posix_trace_set_filter()` function in IEEE Std 1003.1-2001 (see `posix_trace_set_filter()`), selects, before or during tracing, the set of trace event types to be filtered out. It should be possible also (as OSF suggested in their ETAP trace specifications) to select the kernel trace event types to be traced in a system-wide fashion. These two functionalities are called the pre-filtering of trace event types.

The restriction on the actual type used for the `trace_event_set_t` type is intended to guarantee that these objects can always be assigned, have their address taken, and be passed by value as parameters. It is not intended that this type be a structure including pointers to other data structures, as that could impact the portability of applications performing such operations. A reasonable implementation could be a structure containing an array of integer types.

**Filtering of Trace Event Types at Runtime**

It is possible to build this functionality using the `posix_trace_set_filter()` function. A privileged process or a privileged thread can get trace events from the trace stream of another process or thread, and thus specify the type of trace events to record into a file, using implementation-defined methods and interfaces. This functionality, called inline filtering of trace event types, is used for runtime analysis of trace streams.

**Post-Mortem Filtering of Trace Event Types**

The word “post-mortem” is used here to indicate that some unanticipated situation occurs during execution that does not permit a pre or inline filtering of trace events and that it is necessary to record all trace event types to have a chance to discover the problem afterwards. When the program stops, all the trace events recorded previously can be analyzed in order to find the solution. This functionality could be named the post-filtering of trace event types.

**Discussions about Trace Event Type-Filtering**

After long discussions with the parties involved in the process of defining the trace interface, it seems that the sensitivity to the filtering problem is different, but everybody agrees that the level of the overhead introduced during the tracing operation depends on the filtering method elected. If the time that it takes the trace event to be recorded can be neglected, the overhead introduced by the filtering process can be classified as follows:

|                  |                                          |
|------------------|------------------------------------------|
| Pre-filtering    | System and process/thread-level overhead |
| Inline-filtering | Process/thread-level overhead            |
| Post-filtering   | No overhead; done offline                |

The pre-filtering could be named “critical realtime” filtering in the sense that the filtering of trace event type is manageable at the user level so the user can lower to a minimum the filtering overhead at some user selected level of priority for the inline filtering, or delay the filtering to after execution for the post-filtering. The counterpart of this solution is that the size of the trace stream must be sufficient to record all the trace events. The advantage of the pre-filtering is that the utilization of the trace stream is optimized.

Only pre-filtering is defined by IEEE Std 1003.1-2001. However, great care must be taken in specifying pre-filtering, so that it does not impose unacceptable overhead. Moreover, it is necessary to isolate all the functionality relative to the pre-filtering.

The result of this rationale is to define a new option, the Trace Event Filter option, not necessarily implemented in small realtime systems, where system overhead is minimized to the extent possible.



8405 B.2.11.7 Tracing, *pthread* API

8406 The objective to be able to control tracing for individual threads may be in conflict with the  
 8407 efficiency expected in threads with a *contentionscope* attribute of `PTHREAD_SCOPE_PROCESS`.  
 8408 For these threads, context switches from one thread that has tracing enabled to another thread  
 8409 that has tracing disabled may require a kernel call to inform the kernel whether it has to trace  
 8410 system events executed by that thread or not. For this reason, it was proposed that the ability to  
 8411 enable or disable tracing for `PTHREAD_SCOPE_PROCESS` threads be made optional, through  
 8412 the introduction of a Trace Scope Process option. A trace implementation which did not  
 8413 implement the Trace Scope Process option would not honor the tracing-state attribute of a  
 8414 thread with `PTHREAD_SCOPE_PROCESS`; it would, however, honor the tracing-state attribute  
 8415 of a thread with `PTHREAD_SCOPE_SYSTEM`. This proposal was rejected as:

- 8416 1. Removing desired functionality (per-thread trace control)
- 8417 2. Introducing counter-intuitive behavior for the tracing-state attribute
- 8418 3. Mixing logically orthogonal ideas (thread scheduling and thread tracing)
- 8419 [Objective 4]

8420 Finally, to solve this complex issue, this API does not provide *pthread\_gettracingstate()*,  
 8421 *pthread\_settracingstate()*, *pthread\_attr\_gettracingstate()*, and *pthread\_attr\_settracingstate()*  
 8422 interfaces. These interfaces force the thread implementation to add to the weight of the thread  
 8423 and cause a revision of the threads libraries, just to support tracing. Worse yet,  
 8424 *posix\_trace\_event()* must always test this per-thread variable even in the common case where it is  
 8425 not used at all. Per-thread tracing is easy to implement using existing interfaces where necessary;  
 8426 see the following example.

8427 **Example**

```
8428 /* Caution. Error checks omitted */
8429 static pthread_key_t my_key;
8430 static trace_event_id_t my_event_id;
8431 static pthread_once_t my_once = PTHREAD_ONCE_INIT;

8432 void my_init(void)
8433 {
8434 (void) pthread_key_create(&my_key, NULL);
8435 (void) posix_trace_eventid_open("my", &my_event_id);
8436 }

8437 int get_trace_flag(void)
8438 {
8439 pthread_once(&my_once, my_init);
8440 return (pthread_getspecific(my_key) != NULL);
8441 }

8442 void set_trace_flag(int f)
8443 {
8444 pthread_once(&my_once, my_init);
8445 pthread_setspecific(my_key, f? &my_event_id: NULL);
8446 }

8447 fn()
8448 {
8449 if (get_trace_flag())
8450 posix_trace_event(my_event_id, ...)
```

8451 }

8452 The above example does not implement third-party state setting.

8453 Lastly, per-thread tracing works poorly for threads with `PTHREAD_SCOPE_PROCESS`  
8454 contention scope. These “library” threads have minimal interaction with the kernel and would  
8455 have to explicitly set the attributes whenever they are context switched to a new kernel thread in  
8456 order to trace system events. Such state was explicitly avoided in POSIX threads to keep  
8457 `PTHREAD_SCOPE_PROCESS` threads lightweight.

8458 The reason that keeping `PTHREAD_SCOPE_PROCESS` threads lightweight is important is that  
8459 such threads can be used not just for simple multi-processors but also for co-routine style  
8460 programming (such as discrete event simulation) without inventing a new threads paradigm.  
8461 Adding extra runtime cost to thread context switches will make using POSIX threads less  
8462 attractive in these situations.

#### 8463 B.2.11.8 Rationale on Triggering

8464 The ability to start or stop tracing based on the occurrence of specific trace event types has been  
8465 proposed as a parallel to similar functionality appearing in logic analyzers. Such triggering, in  
8466 order to be very useful, should be based not only on the trace event type, but on trace event-  
8467 specific data, including tests of user-specified fields for matching or threshold values.

8468 Such a facility is unnecessary where the buffering of the stream is not a constraint, since such  
8469 checks can be performed offline during post-mortem analysis.

8470 For example, a large system could incorporate a daemon utility to collect the trace records from  
8471 memory buffers and spool them to secondary storage for later analysis. In the instances where  
8472 resources are truly limited, such as embedded applications, the application incorporation of  
8473 application code to test the circumstances of a trace event and call the trace point only if needed  
8474 is usually straightforward.

8475 For performance reasons, the `posix_trace_event()` function should be implemented using a macro,  
8476 so if the trace is inactive, the trace event point calls are latent code and must cost no more than a  
8477 scalar test.

8478 The API proposed in IEEE Std 1003.1-2001 does not include any triggering functionality.

#### 8479 B.2.11.9 Rationale on Timestamp Clock

8480 It has been suggested that the tracing mechanism should include the possibility of specifying the  
8481 clock to be used in timestamping the trace events. When application trace events must be  
8482 correlated to remote trace events, such a facility could provide a global time reference not  
8483 available from a local clock. Further, the application may be driven by timers based on a clock  
8484 different from that used for the timestamp, and the correlation of the trace to those untraced  
8485 timer activities could be an important part of the analysis of the application.

8486 However, the tracing mechanism needs to be fast and just the provision of such an option can  
8487 materially affect its performance. Leaving aside the performance costs of reading some clocks,  
8488 this notion is also ill-defined when kernel trace events are to be traced by two applications  
8489 making use of different tracing clocks. This can even happen within a single application where  
8490 different parts of the application are served by different clocks. Another complication can occur  
8491 when a clock is maintained strictly at the user level and is unavailable at the kernel level.

8492 It is felt that the benefits of a selectable trace clock do not match its costs. Applications that wish  
8493 to correlate clocks other than the default tracing clock can include trace events with sample  
8494 values of those other clocks, allowing correlation of timestamps from the various independent  
8495 clocks. In any case, such a technique would be required when applications are sensitive to

8496 multiple clocks.

#### 8497 *B.2.11.10 Rationale on Different Overrun Conditions*

8498 The analysis of the dynamic behavior of the trace mechanism shows that different overrun  
8499 conditions may occur. The API must provide a means to manage such conditions in a portable  
8500 way.

#### 8501 **Overrun in Trace Streams Initialized with POSIX\_TRACE\_LOOP Policy**

8502 In this case, the user of the trace mechanism is interested in using the trace stream with  
8503 POSIX\_TRACE\_LOOP policy to record trace events continuously, but ideally without losing any  
8504 trace events. The online analyzer process must get the trace events at a mean speed equivalent to  
8505 the recording speed. Should the trace stream become full, a trace stream overrun occurs. This  
8506 condition is detected by getting the status of the active trace stream (function  
8507 *posix\_trace\_get\_status()*) and looking at the member *posix\_stream\_overrun\_status* of the read  
8508 **posix\_stream\_status** structure. In addition, two predefined trace event types are defined:

- 8509 1. The beginning of a trace overflow, to locate the beginning of an overflow when reading a  
8510 trace stream
- 8511 2. The end of a trace overflow, to locate the end of an overflow, when reading a trace stream

8512 As a timestamp is associated with these predefined trace events, it is possible to know the  
8513 duration of the overflow.

#### 8514 **Overrun in Dumping Trace Streams into Trace Logs**

8515 The user lets the trace mechanism dump the trace stream initialized with  
8516 POSIX\_TRACE\_FLUSH policy automatically into a trace log. If the dump operation is slower  
8517 than the recording of trace events, the trace stream can overrun. This condition is detected by  
8518 getting the status of the active trace stream (function *posix\_trace\_get\_status()*) and looking at the  
8519 member *posix\_log\_overrun\_status* of the read **posix\_stream\_status** structure. This overrun  
8520 indicates that the trace mechanism is not able to operate in this mode at this speed. It is the  
8521 responsibility of the user to modify one of the trace parameters (the stream size or the trace  
8522 event type filter, for instance) to avoid such overrun conditions, if overruns are to be prevented.  
8523 The same already predefined trace event types (see **Overrun in Trace Streams Initialized with**  
8524 **POSIX\_TRACE\_LOOP Policy**) are used to detect and to know the duration of an overflow.

#### 8525 **Reading an Active Trace Stream**

8526 Although this trace API allows one to read an active trace stream with log while it is tracing, this  
8527 feature can lead to false overflow origin interpretation: the trace log or the reader of the trace  
8528 stream. Reading from an active trace stream with log is thus non-portable, and has been left  
8529 unspecified.

#### 8530 **B.2.12 Data Types**

8531 The requirement that additional types defined in this section end in “\_t” was prompted by the  
8532 problem of name space pollution. It is difficult to define a type (where that type is not one  
8533 defined by IEEE Std 1003.1-2001) in one header file and use it in another without adding symbols  
8534 to the name space of the program. To allow implementors to provide their own types, all  
8535 conforming applications are required to avoid symbols ending in “\_t”, which permits the  
8536 implementor to provide additional types. Because a major use of types is in the definition of  
8537 structure members, which can (and in many cases must) be added to the structures defined in  
8538 IEEE Std 1003.1-2001, the need for additional types is compelling.

8539 The types, such as **ushort** and **ulong**, which are in common usage, are not defined in  
 8540 IEEE Std 1003.1-2001 (although **ushort\_t** would be permitted as an extension). They can be  
 8541 added to **<sys/types.h>** using a feature test macro (see Section B.2.2.1 (on page 85)). A suggested  
 8542 symbol for these is **\_SYSIII**. Similarly, the types like **u\_short** would probably be best controlled  
 8543 by **\_BSD**.

8544 Some of these symbols may appear in other headers; see Section B.2.2.2 (on page 86).

8545 **dev\_t** This type may be made large enough to accommodate host-locality considerations  
 8546 of networked systems.

8547 This type must be arithmetic. Earlier proposals allowed this to be non-arithmetic  
 8548 (such as a structure) and provided a *samefile()* function for comparison.

8549 **gid\_t** Some implementations had separated **gid\_t** from **uid\_t** before POSIX.1 was  
 8550 completed. It would be difficult for them to coalesce them when it was  
 8551 unnecessary. Additionally, it is quite possible that user IDs might be different from  
 8552 group IDs because the user ID might wish to span a heterogeneous network,  
 8553 where the group ID might not.

8554 For current implementations, the cost of having a separate **gid\_t** will be only  
 8555 lexical.

8556 **mode\_t** This type was chosen so that implementations could choose the appropriate  
 8557 integer type, and for compatibility with the ISO C standard. 4.3 BSD uses  
 8558 **unsigned short** and the SVID uses **ushort**, which is the same. Historically, only the  
 8559 low-order sixteen bits are significant.

8560 **nlink\_t** This type was introduced in place of **short** for *st\_nlink* (see the **<sys/stat.h>** header)  
 8561 in response to an objection that **short** was too small.

8562 **off\_t** This type is used only in *lseek()*, *fcntl()*, and **<sys/stat.h>**. Many implementations  
 8563 would have difficulties if it were defined as anything other than **long**. Requiring  
 8564 an integer type limits the capabilities of *lseek()* to four gigabytes. The ISO C  
 8565 standard supplies routines that use larger types; see *fgetpos()* and *fsetpos()*. XSI-  
 8566 conformant systems provide the *fseeko()* and *ftello()* functions that use larger  
 8567 types.

8568 **pid\_t** The inclusion of this symbol was controversial because it is tied to the issue of the  
 8569 representation of a process ID as a number. From the point of view of a  
 8570 conforming application, process IDs should be “magic cookies”<sup>1</sup> that are produced  
 8571 by calls such as *fork()*, used by calls such as *waitpid()* or *kill()*, and not otherwise  
 8572 analyzed (except that the sign is used as a flag for certain operations).

8573 The concept of a {PID\_MAX} value interacted with this in early proposals. Treating  
 8574 process IDs as an opaque type both removes the requirement for {PID\_MAX} and  
 8575 allows systems to be more flexible in providing process IDs that span a large range  
 8576 of values, or a small one.

8577 Since the values in **uid\_t**, **gid\_t**, and **pid\_t** will be numbers generally, and  
 8578 potentially both large in magnitude and sparse, applications that are based on

8579

---

8580 1. An historical term meaning: “An opaque object, or token, of determinate size, whose significance is known only to the entity  
 8581 which created it. An entity receiving such a token from the generating entity may only make such use of the ‘cookie’ as is defined  
 8582 and permitted by the supplying entity.”

|      |                |                                                                                                                                    |
|------|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| 8583 |                | arrays of objects of this type are unlikely to be fully portable in any case. Solutions                                            |
| 8584 |                | that treat them as magic cookies will be portable.                                                                                 |
| 8585 |                | {CHILD_MAX} precludes the possibility of a “toy implementation”, where there                                                       |
| 8586 |                | would only be one process.                                                                                                         |
| 8587 | <b>ssize_t</b> | This is intended to be a signed analog of <b>size_t</b> . The wording is such that an                                              |
| 8588 |                | implementation may either choose to use a longer type or simply to use the signed                                                  |
| 8589 |                | version of the type that underlies <b>size_t</b> . All functions that return <b>ssize_t</b> ( <i>read()</i>                        |
| 8590 |                | and <i>write()</i> ) describe as “implementation-defined” the result of an input exceeding                                         |
| 8591 |                | {SSIZE_MAX}. It is recognized that some implementations might have <b>ints</b> that                                                |
| 8592 |                | are smaller than <b>size_t</b> . A conforming application would be constrained not to                                              |
| 8593 |                | perform I/O in pieces larger than {SSIZE_MAX}, but a conforming application                                                        |
| 8594 |                | using extensions would be able to use the full range if the implementation                                                         |
| 8595 |                | provided an extended range, while still having a single type-compatible interface.                                                 |
| 8596 |                | The symbols <b>size_t</b> and <b>ssize_t</b> are also required in <b>&lt;unistd.h&gt;</b> to minimize the                          |
| 8597 |                | changes needed for calls to <i>read()</i> and <i>write()</i> . Implementors are reminded that it                                   |
| 8598 |                | must be possible to include both <b>&lt;sys/types.h&gt;</b> and <b>&lt;unistd.h&gt;</b> in the same                                |
| 8599 |                | program (in either order) without error.                                                                                           |
| 8600 | <b>uid_t</b>   | Before the addition of this type, the data types used to represent these values                                                    |
| 8601 |                | varied throughout early proposals. The <b>&lt;sys/stat.h&gt;</b> header defined these values as                                    |
| 8602 |                | type <b>short</b> , the <b>&lt;passwd.h&gt;</b> file (now <b>&lt;pwd.h&gt;</b> and <b>&lt;grp.h&gt;</b> ) used an <b>int</b> , and |
| 8603 |                | <i>getuid()</i> returned an <b>int</b> . In response to a strong objection to the inconsistent                                     |
| 8604 |                | definitions, all the types were switched to <b>uid_t</b> .                                                                         |
| 8605 |                | In practice, those historical implementations that use varying types of this sort can                                              |
| 8606 |                | typedef <b>uid_t</b> to <b>short</b> with no serious consequences.                                                                 |
| 8607 |                | The problem associated with this change concerns object compatibility after                                                        |
| 8608 |                | structure size changes. Since most implementations will define <b>uid_t</b> as a short, the                                        |
| 8609 |                | only substantive change will be a reduction in the size of the <b>passwd</b> structure.                                            |
| 8610 |                | Consequently, implementations with an overriding concern for object                                                                |
| 8611 |                | compatibility can pad the structure back to its current size. For that reason, this                                                |
| 8612 |                | problem was not considered critical enough to warrant the addition of a separate                                                   |
| 8613 |                | type to POSIX.1.                                                                                                                   |
| 8614 |                | The types <b>uid_t</b> and <b>gid_t</b> are magic cookies. There is no {UID_MAX} defined by                                        |
| 8615 |                | POSIX.1, and no structure imposed on <b>uid_t</b> and <b>gid_t</b> other than that they be                                         |
| 8616 |                | positive arithmetic types. (In fact, they could be <b>unsigned char</b> .) There is no                                             |
| 8617 |                | maximum or minimum specified for the number of distinct user or group IDs.                                                         |

## 8618 **B.3 System Interfaces**

8619 See the RATIONALE sections on the individual reference pages.

### 8620 **B.3.1 Examples for Spawn**

8621 The following long examples are provided in the Rationale (Informative) volume of  
8622 IEEE Std 1003.1-2001 as a supplement to the reference page for *posix\_spawn()*.

#### 8623 **Example Library Implementation of Spawn**

8624 The *posix\_spawn()* or *posix\_spawnnp()* functions provide the following:

- 8625 • Simply start a process executing a process image. This is the simplest application for process  
8626 creation, and it may cover most executions of *fork()*.
- 8627 • Support I/O redirection, including pipes.
- 8628 • Run the child under a user and group ID in the domain of the parent.
- 8629 • Run the child at any priority in the domain of the parent.

8630 The *posix\_spawn()* or *posix\_spawnnp()* functions do not cover every possible use of the *fork()*  
8631 function, but they do span the common applications: typical use by a shell and a login utility.

8632 The price for an application is that before it calls *posix\_spawn()* or *posix\_spawnnp()*, the parent  
8633 must adjust to a state that *posix\_spawn()* or *posix\_spawnnp()* can map to the desired state for the  
8634 child. Environment changes require the parent to save some of its state and restore it afterwards.  
8635 The effective behavior of a successful invocation of *posix\_spawn()* is as if the operation were  
8636 implemented with POSIX operations as follows:

```

8637 #include <sys/types.h>
8638 #include <stdlib.h>
8639 #include <stdio.h>
8640 #include <unistd.h>
8641 #include <sched.h>
8642 #include <fcntl.h>
8643 #include <signal.h>
8644 #include <errno.h>
8645 #include <string.h>
8646 #include <signal.h>

8647 /* #include <spawn.h> */
8648 /*****
8649 /* Things that could be defined in spawn.h */
8650 *****/
8651 typedef struct
8652 {
8653 short posix_attr_flags;
8654 #define POSIX_SPAWN_SETPGROUP 0x1
8655 #define POSIX_SPAWN_SETSIGMASK 0x2
8656 #define POSIX_SPAWN_SETSIGDEF 0x4
8657 #define POSIX_SPAWN_SETSCHEDULER 0x8
8658 #define POSIX_SPAWN_SETSCHEDPARAM 0x10
8659 #define POSIX_SPAWN_RESETIDS 0x20
8660 pid_t posix_attr_pgroup;
8661 sigset_t posix_attr_sigmask;
8662 sigset_t posix_attr_sigdefault;

```

```

8663 int posix_attr_schedpolicy;
8664 struct sched_param posix_attr_schedparam;
8665 } posix_spawnattr_t;

8666 typedef char *posix_spawn_file_actions_t;

8667 int posix_spawn_file_actions_init(
8668 posix_spawn_file_actions_t *file_actions);
8669 int posix_spawn_file_actions_destroy(
8670 posix_spawn_file_actions_t *file_actions);
8671 int posix_spawn_file_actions_addclose(
8672 posix_spawn_file_actions_t *file_actions, int fildes);
8673 int posix_spawn_file_actions_adddup2(
8674 posix_spawn_file_actions_t *file_actions, int fildes,
8675 int newfildes);
8676 int posix_spawn_file_actions_addopen(
8677 posix_spawn_file_actions_t *file_actions, int fildes,
8678 const char *path, int oflag, mode_t mode);
8679 int posix_spawnattr_init(posix_spawnattr_t *attr);
8680 int posix_spawnattr_destroy(posix_spawnattr_t *attr);
8681 int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
8682 short *lags);
8683 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
8684 int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
8685 pid_t *pgroup);
8686 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
8687 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
8688 int *schedpolicy);
8689 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
8690 int schedpolicy);
8691 int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
8692 struct sched_param *schedparam);
8693 int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
8694 const struct sched_param *schedparam);
8695 int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
8696 sigset_t *sigmask);
8697 int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
8698 const sigset_t *sigmask);
8699 int posix_spawnattr_getdefault(const posix_spawnattr_t *attr,
8700 sigset_t *sigdefault);
8701 int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
8702 const sigset_t *sigdefault);
8703 int posix_spawn(pid_t *pid, const char *path,
8704 const posix_spawn_file_actions_t *file_actions,
8705 const posix_spawnattr_t *attrp, char *const argv[],
8706 char *const envp[]);
8707 int posix_spawnnp(pid_t *pid, const char *file,
8708 const posix_spawn_file_actions_t *file_actions,
8709 const posix_spawnattr_t *attrp, char *const argv[],
8710 char *const envp[]);

8711 /*****
8712 /* Example posix_spawn() library routine */
8713 /*****

```

```

8714 int posix_spawn(pid_t *pid,
8715 const char *path,
8716 const posix_spawn_file_actions_t *file_actions,
8717 const posix_spawnattr_t *attrp,
8718 char *const argv[],
8719 char *const envp[])
8720 {
8721 /* Create process */
8722 if ((*pid = fork()) == (pid_t) 0)
8723 {
8724 /* This is the child process */
8725 /* Worry about process group */
8726 if (attrp->posix_attr_flags & POSIX_SPAWN_SETPGROUP)
8727 {
8728 /* Override inherited process group */
8729 if (setpgid(0, attrp->posix_attr_pgroup) != 0)
8730 {
8731 /* Failed */
8732 exit(127);
8733 }
8734 }
8735 /* Worry about thread signal mask */
8736 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGMASK)
8737 {
8738 /* Set the signal mask (can't fail) */
8739 sigprocmask(SIG_SETMASK, &attrp->posix_attr_sigmask, NULL);
8740 }
8741 /* Worry about resetting effective user and group IDs */
8742 if (attrp->posix_attr_flags & POSIX_SPAWN_RESETPIDS)
8743 {
8744 /* None of these can fail for this case. */
8745 setuid(getuid());
8746 setgid(getgid());
8747 }
8748 /* Worry about defaulted signals */
8749 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGDEF)
8750 {
8751 struct sigaction deflt;
8752 sigset_t all_signals;
8753
8754 int s;
8755
8756 /* Construct default signal action */
8757 deflt.sa_handler = SIG_DFL;
8758 deflt.sa_flags = 0;
8759
8760 /* Construct the set of all signals */
8761 sigfillset(&all_signals);
8762
8763 /* Loop for all signals */
8764 for (s = 0; sigismember(&all_signals, s); s++)
8765 {
8766 /* Signal to be defaulted? */

```



```

8763 if (sigismember(&attrp->posix_attr_sigdefault, s))
8764 {
8765 /* Yes; default this signal */
8766 if (sigaction(s, &deflt, NULL) == -1)
8767 {
8768 /* Failed */
8769 exit(127);
8770 }
8771 }
8772 }
8773 }

8774 /* Worry about the fds if they are to be mapped */
8775 if (file_actions != NULL)
8776 {
8777 /* Loop for all actions in object file_actions */
8778 /* (implementation dives beneath abstraction) */
8779 char *p = *file_actions;

8780 while (*p != '\0')
8781 {
8782 if (strncmp(p, "close(", 6) == 0)
8783 {
8784 int fd;

8785 if (sscanf(p + 6, "%d", &fd) != 1)
8786 {
8787 exit(127);
8788 }
8789 if (close(fd) == -1)
8790 exit(127);
8791 }
8792 else if (strncmp(p, "dup2(", 5) == 0)
8793 {
8794 int fd, newfd;

8795 if (sscanf(p + 5, "%d,%d", &fd, &newfd) != 2)
8796 {
8797 exit(127);
8798 }
8799 if (dup2(fd, newfd) == -1)
8800 exit(127);
8801 }
8802 else if (strncmp(p, "open(", 5) == 0)
8803 {
8804 int fd, oflag;
8805 mode_t mode;
8806 int tempfd;
8807 char path[1000]; /* Should be dynamic */
8808 char *q;

8809 if (sscanf(p + 5, "%d,", &fd) != 1)
8810 {
8811 exit(127);
8812 }

```

```

8813 p = strchr(p, ',') + 1;
8814 q = strchr(p, '*');
8815 if (q == NULL)
8816 exit(127);
8817 strncpy(path, p, q - p);
8818 path[q - p] = '\0';
8819 if (sscanf(q + 1, "%o,%o", &oflag, &mode) != 2)
8820 {
8821 exit(127);
8822 }
8823 if (close(fd) == -1)
8824 {
8825 if (errno != EBADF)
8826 exit(127);
8827 }
8828 tempfd = open(path, oflag, mode);
8829 if (tempfd == -1)
8830 exit(127);
8831 if (tempfd != fd)
8832 {
8833 if (dup2(tempfd, fd) == -1)
8834 {
8835 exit(127);
8836 }
8837 if (close(tempfd) == -1)
8838 {
8839 exit(127);
8840 }
8841 }
8842 }
8843 else
8844 {
8845 exit(127);
8846 }
8847 p = strchr(p, ')') + 1;
8848 }
8849
8850 /* Worry about setting new scheduling policy and parameters */
8851 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDULER)
8852 {
8853 if (sched_setscheduler(0, attrp->posix_attr_schedpolicy,
8854 &attrp->posix_attr_schedparam) == -1)
8855 {
8856 exit(127);
8857 }
8858 }
8859
8859 /* Worry about setting only new scheduling parameters */
8860 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDPARAM)
8861 {
8862 if (sched_setparam(0, &attrp->posix_attr_schedparam) == -1)
8863 {

```

```

8864 exit(127);
8865 }
8866 }

8867 /* Now execute the program at path */
8868 /* Any fd that still has FD_CLOEXEC set will be closed */
8869 execve(path, argv, envp);
8870 exit(127); /* exec failed */
8871 }
8872 else
8873 {
8874 /* This is the parent (calling) process */
8875 if (*pid == (pid_t) - 1)
8876 return errno;
8877 return 0;
8878 }
8879 }

8880 /******
8881 /* Here is a crude but effective implementation of the */
8882 /* file action object operators which store actions as */
8883 /* concatenated token-separated strings. */
8884 /******
8885 /* Create object with no actions. */
8886 int posix_spawn_file_actions_init(
8887 posix_spawn_file_actions_t *file_actions)
8888 {
8889 *file_actions = malloc(sizeof(char));
8890 if (*file_actions == NULL)
8891 return ENOMEM;
8892 strcpy(*file_actions, "");
8893 return 0;
8894 }

8895 /* Free object storage and make invalid. */
8896 int posix_spawn_file_actions_destroy(
8897 posix_spawn_file_actions_t *file_actions)
8898 {
8899 free(*file_actions);
8900 *file_actions = NULL;
8901 return 0;
8902 }

8903 /* Add a new action string to object. */
8904 static int add_to_file_actions(
8905 posix_spawn_file_actions_t *file_actions, char *new_action)
8906 {
8907 *file_actions = realloc
8908 (*file_actions, strlen(*file_actions) + strlen(new_action) + 1);
8909 if (*file_actions == NULL)
8910 return ENOMEM;
8911 strcat(*file_actions, new_action);
8912 return 0;
8913 }

```

```

8914 /* Add a close action to object. */
8915 int posix_spawn_file_actions_addclose(
8916 posix_spawn_file_actions_t *file_actions, int fildes)
8917 {
8918 char temp[100];
8919
8919 sprintf(temp, "close(%d)", fildes);
8920 return add_to_file_actions(file_actions, temp);
8921 }
8922
8922 /* Add a dup2 action to object. */
8923 int posix_spawn_file_actions_adddup2(
8924 posix_spawn_file_actions_t *file_actions, int fildes,
8925 int newfildes)
8926 {
8927 char temp[100];
8928
8928 sprintf(temp, "dup2(%d,%d)", fildes, newfildes);
8929 return add_to_file_actions(file_actions, temp);
8930 }
8931
8931 /* Add an open action to object. */
8932 int posix_spawn_file_actions_addopen(
8933 posix_spawn_file_actions_t *file_actions, int fildes,
8934 const char *path, int oflag, mode_t mode)
8935 {
8936 char temp[100];
8937
8937 sprintf(temp, "open(%d,%s*%o,%o)", fildes, path, oflag, mode);
8938 return add_to_file_actions(file_actions, temp);
8939 }
8940
8940 /*****
8941 /* Here is a crude but effective implementation of the */
8942 /* spawn attributes object functions which manipulate */
8943 /* the individual attributes. */
8944 *****/
8945 /* Initialize object with default values. */
8946 int posix_spawnattr_init(posix_spawnattr_t *attr)
8947 {
8948 attr->posix_attr_flags = 0;
8949 attr->posix_attr_pgroup = 0;
8950 /* Default value of signal mask is the parent's signal mask; */
8951 /* other values are also allowed */
8952 sigprocmask(0, NULL, &attr->posix_attr_sigmask);
8953 sigemptyset(&attr->posix_attr_sigdefault);
8954 /* Default values of scheduling attr inherited from the parent; */
8955 /* other values are also allowed */
8956 attr->posix_attr_schedpolicy = sched_getscheduler(0);
8957 sched_getparam(0, &attr->posix_attr_schedparam);
8958 return 0;
8959 }
8960
8960 int posix_spawnattr_destroy(posix_spawnattr_t *attr)
8961 {
8962 /* No action needed */

```

```
8963 return 0;
8964 }
8965 int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
8966 short *flags)
8967 {
8968 *flags = attr->posix_attr_flags;
8969 return 0;
8970 }
8971 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags)
8972 {
8973 attr->posix_attr_flags = flags;
8974 return 0;
8975 }
8976 int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
8977 pid_t *pgroup)
8978 {
8979 *pgroup = attr->posix_attr_pgroup;
8980 return 0;
8981 }
8982 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup)
8983 {
8984 attr->posix_attr_pgroup = pgroup;
8985 return 0;
8986 }
8987 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
8988 int *schedpolicy)
8989 {
8990 *schedpolicy = attr->posix_attr_schedpolicy;
8991 return 0;
8992 }
8993 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
8994 int schedpolicy)
8995 {
8996 attr->posix_attr_schedpolicy = schedpolicy;
8997 return 0;
8998 }
8999 int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
9000 struct sched_param *schedparam)
9001 {
9002 *schedparam = attr->posix_attr_schedparam;
9003 return 0;
9004 }
9005 int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
9006 const struct sched_param *schedparam)
9007 {
9008 attr->posix_attr_schedparam = *schedparam;
9009 return 0;
9010 }
```

```

9011 int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
9012 sigset_t *sigmask)
9013 {
9014 *sigmask = attr->posix_attr_sigmask;
9015 return 0;
9016 }
9017
9018 int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
9019 const sigset_t *sigmask)
9020 {
9021 attr->posix_attr_sigmask = *sigmask;
9022 return 0;
9023 }
9024
9025 int posix_spawnattr_getsigdefault(const posix_spawnattr_t *attr,
9026 sigset_t *sigdefault)
9027 {
9028 *sigdefault = attr->posix_attr_sigdefault;
9029 return 0;
9030 }
9031
9032 int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
9033 const sigset_t *sigdefault)
9034 {
9035 attr->posix_attr_sigdefault = *sigdefault;
9036 return 0;
9037 }
9038
9039

```

### I/O Redirection with Spawn

I/O redirection with *posix\_spawn()* or *posix\_spawnnp()* is accomplished by crafting a *file\_actions* argument to effect the desired redirection. Such a redirection follows the general outline of the following example:

```

9039 /* To redirect new standard output (fd 1) to a file, */
9040 /* and redirect new standard input (fd 0) from my fd socket_pair[1], */
9041 /* and close my fd socket_pair[0] in the new process. */
9042 posix_spawn_file_actions_t file_actions;
9043 posix_spawn_file_actions_init(&file_actions);
9044 posix_spawn_file_actions_addopen(&file_actions, 1, "newout", ...);
9045 posix_spawn_file_actions_dup2(&file_actions, socket_pair[1], 0);
9046 posix_spawn_file_actions_close(&file_actions, socket_pair[0]);
9047 posix_spawn_file_actions_close(&file_actions, socket_pair[1]);
9048 posix_spawn(..., &file_actions, ...);
9049 posix_spawn_file_actions_destroy(&file_actions);

```

**9050        Spawning a Process Under a New User ID**

9051        Spawning a process under a new user ID follows the outline shown in the following example:

```
9052 Save = getuid();
9053 setuid(newid);
9054 posix_spawn(...);
9055 setuid(Save);
```







# *Rationale (Informative)*

9057

9058

## **Part C:**

9059

## **Shell and Utilities**

9060

*The Open Group*

9061

*The Institute of Electrical and Electronics Engineers, Inc.*



# Rationale for Shell and Utilities

9062

## 9063 C.1 Introduction

### 9064 C.1.1 Scope

9065 Refer to Section A.1.1 (on page 3).

### 9066 C.1.2 Conformance

9067 Refer to Section A.2 (on page 9).

### 9068 C.1.3 Normative References

9069 There is no additional rationale provided for this section.

### 9070 C.1.4 Change History

9071 The change history is provided as an informative section, to track changes from previous issues  
9072 of IEEE Std 1003.1-2001.

9073 The following sections describe changes made to the Shell and Utilities volume of  
9074 IEEE Std 1003.1-2001 since Issue 5 of the base document. The CHANGE HISTORY section for  
9075 each utility describes technical changes made to that utility from Issue 5. Changes between  
9076 earlier issues of the base document and Issue 5 are not included.

9077 The change history between Issue 5 and Issue 6 also lists the changes since the  
9078 ISO POSIX-2: 1993 standard.

#### 9079 Changes from Issue 5 to Issue 6 (IEEE Std 1003.1-2001)

9080 The following list summarizes the major changes that were made in the Shell and Utilities  
9081 volume of IEEE Std 1003.1-2001 from Issue 5 to Issue 6:

- 9082 • This volume of IEEE Std 1003.1-2001 is extensively revised so that it can be both an IEEE  
9083 POSIX Standard and an Open Group Technical Standard.
- 9084 • The terminology has been reworked to meet the style requirements.
- 9085 • Shading notation and margin codes are introduced for identification of options within the  
9086 volume.
- 9087 • This volume of IEEE Std 1003.1-2001 is updated to mandate support of FIPS 151-2. The  
9088 following changes were made:
  - 9089 — Support is mandated for the capabilities associated with the following symbolic  
9090 constants:

9091        \_POSIX\_CHOWN\_RESTRICTED  
9092        \_POSIX\_JOB\_CONTROL  
9093        \_POSIX\_SAVED\_IDS

- 9094 — In the environment for the login shell, the environment variables *LOGNAME* and *HOME*  
9095 shall be defined and have the properties described in the Base Definitions volume of

- 9096 IEEE Std 1003.1-2001, Chapter 7, Locale.
- 9097 • This volume of IEEE Std 1003.1-2001 is updated to align with some features of the Single  
9098 UNIX Specification.
- 9099 • A new section on Utility Limits is added.
- 9100 • A section on the Relationships to Other Documents is added.
- 9101 • Concepts and definitions have been moved to a separate volume.
- 9102 • A RATIONALE section is added to each reference page.
- 9103 • The *c99* utility is added as a replacement for *c89*, which is withdrawn in this issue.
- 9104 • IEEE Std 1003.2d-1994 is incorporated, adding the *qalter*, *qdel*, *qhold*, *qmove*, *qmsg*, *qrerun*, *qrls*,  
9105 *qselect*, *qsig*, *qstat*, and *qsub* utilities.
- 9106 • IEEE P1003.2b draft standard is incorporated, making extensive updates and adding the *iconv*  
9107 utility.
- 9108 • IEEE PASC Interpretations are applied.
- 9109 • The Open Group's corrigenda and resolutions are applied.

#### 9110 **New Features in Issue 6**

9111 The following table lists the new utilities introduced since the ISO POSIX-2:1993 standard (as  
9112 modified by IEEE Std 1003.2d-1994). Apart from the *c99* and *iconv* utilities, these are all part of  
9113 the XSI extension.

| New Utilities in Issue 6 |                 |               |              |             |                   |               |             |
|--------------------------|-----------------|---------------|--------------|-------------|-------------------|---------------|-------------|
| 9116 <i>admin</i>        | <i>compress</i> | <i>gencat</i> | <i>ipcrm</i> | <i>nl</i>   | <i>tsort</i>      | <i>unlink</i> | <i>val</i>  |
| 9117 <i>c99</i>          | <i>cxref</i>    | <i>get</i>    | <i>ipcs</i>  | <i>prs</i>  | <i>ulimit</i>     | <i>uucp</i>   | <i>what</i> |
| 9118 <i>cal</i>          | <i>delta</i>    | <i>hash</i>   | <i>link</i>  | <i>sact</i> | <i>uncompress</i> | <i>uustat</i> | <i>zcat</i> |
| 9119 <i>cflow</i>        | <i>fuser</i>    | <i>iconv</i>  | <i>m4</i>    | <i>sccs</i> | <i>unget</i>      | <i>uux</i>    |             |

#### 9120 **C.1.5 Terminology**

9121 Refer to Section A.1.4 (on page 5).

#### 9122 **C.1.6 Definitions**

9123 Refer to Section A.3 (on page 13).

#### 9124 **C.1.7 Relationship to Other Documents**

##### 9125 *C.1.7.1 System Interfaces*

9126 It has been pointed out that the Shell and Utilities volume of IEEE Std 1003.1-2001 assumes that  
9127 a great deal of functionality from the System Interfaces volume of IEEE Std 1003.1-2001 is  
9128 present, but never states exactly how much (and strictly does not need to since both are  
9129 mandated on a conforming system). This section is an attempt to clarify the assumptions.

9130 **File Removal**

9131 This is intended to be a summary of the *unlink()* and *rmdir()* requirements. Note that it is  
9132 possible using the *unlink()* function for item 4. to occur.

9133 *C.1.7.2 Concepts Derived from the ISO C Standard*

9134 This section was introduced to address the issue that there was insufficient detail presented by  
9135 such utilities as *awk* or *sh* about their procedural control statements and their methods of  
9136 performing arithmetic functions.

9137 The ISO C standard was selected as a model because most historical implementations of the  
9138 standard utilities were written in C. Thus, it was more likely that they would act in the desired  
9139 manner without modification.

9140 Using the ISO C standard is primarily a notational convenience so that the many procedural  
9141 languages in the Shell and Utilities volume of IEEE Std 1003.1-2001 would not have to be  
9142 rigorously described in every aspect. Its selection does not require that the standard utilities be  
9143 written in Standard C; they could be written in Common Usage C, Ada, Pascal, assembler  
9144 language, or anything else.

9145 The sizes of the various numeric values refer to C-language data types that are allowed to be  
9146 different sizes by the ISO C standard. Thus, like a C-language application, a shell application  
9147 cannot rely on their exact size. However, it can rely on their minimum sizes expressed in the  
9148 ISO C standard, such as {LONG\_MAX} for a **long** type.

9149 The behavior on overflow is undefined for ISO C standard arithmetic. Therefore, the standard  
9150 utilities can use “bignum” representation for integers so that there is no fixed maximum unless  
9151 otherwise stated in the utility description. Similarly, standard utilities can use infinite-precision  
9152 representations for floating-point arithmetic, as long as these representations exceed the ISO C  
9153 standard requirements.

9154 This section addresses only the issue of semantics; it is not intended to specify syntax. For  
9155 example, the ISO C standard requires that 0L be recognized as an integer constant equal to zero,  
9156 but utilities such as *awk* and *sh* are not required to recognize 0L (though they are allowed to, as  
9157 an extension).

9158 The ISO C standard requires that a C compiler must issue a diagnostic for constants that are too  
9159 large to represent. Most standard utilities are not required to issue these diagnostics; for  
9160 example, the command:

9161 `diff -C 2147483648 file1 file2`

9162 has undefined behavior, and the *diff* utility is not required to issue a diagnostic even if the  
9163 number 2 147 483 648 cannot be represented.

9164 **C.1.8 Portability**

9165 Refer to Section A.1.5 (on page 8).

9166 *C.1.8.1 Codes*

9167 Refer to Section A.1.5.1 (on page 8).

### 9168 C.1.9 Utility Limits

9169 This section grew out of an idea that originated with the original POSIX.1, in the tables of system  
 9170 limits for the `sysconf()` and `pathconf()` functions. The idea being that a conforming application  
 9171 can be written to use the most restrictive values that a minimal system can provide, but it should  
 9172 not have to. The values provided represent compromises so that some vendors can use  
 9173 historically limited versions of UNIX system utilities. They are the highest values that a strictly  
 9174 conforming application can assume, given no other information.

9175 However, by using the `getconf` utility or the `sysconf()` function, the elegant application can be  
 9176 tailored to more liberal values on some of the specific instances of specific implementations.

9177 There is no explicitly stated requirement that an implementation provide finite limits for any of  
 9178 these numeric values; the implementation is free to provide essentially unbounded capabilities  
 9179 (where it makes sense), stopping only at reasonable points such as `{ULONG_MAX}` (from the  
 9180 ISO C standard). Therefore, applications desiring to tailor themselves to the values on a  
 9181 particular implementation need to be ready for possibly huge values; it may not be a good idea  
 9182 to allocate blindly a buffer for an input line based on the value of `{LINE_MAX}`, for instance.  
 9183 However, unlike the System Interfaces volume of IEEE Std 1003.1-2001, there is no set of limits  
 9184 that return a special indication meaning “unbounded”. The implementation should always  
 9185 return an actual number, even if the number is very large.

9186 The statement:

9187 “It is not guaranteed that the application ...”

9188 is an indication that many of these limits are designed to ensure that implementors design their  
 9189 utilities without arbitrary constraints related to unimaginative programming. There are certainly  
 9190 conditions under which combinations of options can cause failures that would not render an  
 9191 implementation non-conforming. For example, `{EXPR_NEST_MAX}` and `{ARG_MAX}` could  
 9192 collide when expressions are large; combinations of `{BC_SCALE_MAX}` and `{BC_DIM_MAX}`  
 9193 could exceed virtual memory.

9194 In the Shell and Utilities volume of IEEE Std 1003.1-2001, the notion of a limit being guaranteed  
 9195 for the process lifetime, as it is in the System Interfaces volume of IEEE Std 1003.1-2001, is not as  
 9196 useful to a shell script. The `getconf` utility is probably a process itself, so the guarantee would be  
 9197 without value. Therefore, the Shell and Utilities volume of IEEE Std 1003.1-2001 requires the  
 9198 guarantee to be for the session lifetime. This will mean that many vendors will either return very  
 9199 conservative values or possibly implement `getconf` as a built-in.

9200 It may seem confusing to have limits that apply only to a single utility grouped into one global  
 9201 section. However, the alternative, which would be to disperse them out into their utility  
 9202 description sections, would cause great difficulty when `sysconf()` and `getconf` were described.  
 9203 Therefore, the standard developers chose the global approach.

9204 Each language binding could provide symbol names that are slightly different from those shown  
 9205 here. For example, the C-Language Binding option adds a leading underscore to the symbols as a  
 9206 prefix.

9207 The following comments describe selection criteria for the symbols and their values:

9208 `{ARG_MAX}`

9209 This is defined by the System Interfaces volume of IEEE Std 1003.1-2001. Unfortunately, it is  
 9210 very difficult for a conforming application to deal with this value, as it does not know how  
 9211 much of its argument space is being consumed by the environment variables of the user.

9212 {BC\_BASE\_MAX}  
 9213 {BC\_DIM\_MAX}  
 9214 {BC\_SCALE\_MAX}  
 9215 These were originally one value, {BC\_SCALE\_MAX}, but it was unreasonable to link all  
 9216 three concepts into one limit.

9217 {CHILD\_MAX}  
 9218 This is defined by the System Interfaces volume of IEEE Std 1003.1-2001.

9219 {COLL\_WEIGHTS\_MAX}  
 9220 The weights assigned to **order** can be considered as “passes” through the collation  
 9221 algorithm.

9222 {EXPR\_NEST\_MAX}  
 9223 The value for expression nesting was borrowed from the ISO C standard.

9224 {LINE\_MAX}  
 9225 This is a global limit that affects all utilities, unless otherwise noted. The {MAX\_CANON}  
 9226 value from the System Interfaces volume of IEEE Std 1003.1-2001 may further limit input  
 9227 lines from terminals. The {LINE\_MAX} value was the subject of much debate and is a  
 9228 compromise between those who wished to have unlimited lines and those who understood  
 9229 that many historical utilities were written with fixed buffers. Frequently, utility writers  
 9230 selected the UNIX system constant BUFSIZ to allocate these buffers; therefore, some utilities  
 9231 were limited to 512 bytes for I/O lines, while others achieved 4096 bytes or greater.

9232 It should be noted that {LINE\_MAX} applies only to input line length; there is no  
 9233 requirement in IEEE Std 1003.1-2001 that limits the length of output lines. Utilities such as  
 9234 *awk*, *sed*, and *paste* could theoretically construct lines longer than any of the input lines they  
 9235 received, depending on the options used or the instructions from the application. They are  
 9236 not required to truncate their output to {LINE\_MAX}. It is the responsibility of the  
 9237 application to deal with this. If the output of one of those utilities is to be piped into another  
 9238 of the standard utilities, line length restrictions will have to be considered; the *fold* utility,  
 9239 among others, could be used to ensure that only reasonable line lengths reach utilities or  
 9240 applications.

9241 {LINK\_MAX}  
 9242 This is defined by the System Interfaces volume of IEEE Std 1003.1-2001.

9243 {MAX\_CANON}  
 9244 {MAX\_INPUT}  
 9245 {NAME\_MAX}  
 9246 {NGROUPS\_MAX}  
 9247 {OPEN\_MAX}  
 9248 {PATH\_MAX}  
 9249 {PIPE\_BUF}  
 9250 These limits are defined by the System Interfaces volume of IEEE Std 1003.1-2001. Note that  
 9251 the byte lengths described by some of these values continue to represent bytes, even if the  
 9252 applicable character set uses a multi-byte encoding.

9253 {RE\_DUP\_MAX}  
 9254 The value selected is consistent with historical practice. Although the name implies that it  
 9255 applies to all REs, only BREs use the interval notation `\{m,n\}` addressed by this limit.

9256 {POSIX2\_SYMLINKS}  
 9257 The {POSIX2\_SYMLINKS} variable indicates that the underlying operating system supports  
 9258 the creation of symbolic links in specific directories. Many of the utilities defined in  
 9259 IEEE Std 1003.1-2001 that deal with symbolic links do not depend on this value. For

example, a utility that follows symbolic links (or does not, as the case may be) will only be affected by a symbolic link if it encounters one. Presumably, a file system that does not support symbolic links will not contain any. This variable does affect such utilities as *ln -s* and *pax* that attempt to create symbolic links.

{POSIX2\_SYMLINKS} was developed even though there is no comparable configuration value for the system interfaces.

There are different limits associated with command lines and input to utilities, depending on the method of invocation. In the case of a C program *exec*-ing a utility, {ARG\_MAX} is the underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE\_MAX} limits the length of lines the shell is required to process, and {ARG\_MAX} will still be a limit. If a user is entering a command on a terminal to the shell, requesting that it invoke the utility, {MAX\_INPUT} may restrict the length of the line that can be given to the shell to a value below {LINE\_MAX}.

When an option is supported, *getconf* returns a value of 1. For example, when C development is supported:

```
if ["$(getconf POSIX2_C_DEV)" -eq 1]; then
 echo C supported
fi
```

The *sysconf()* function in the C-Language Binding option would return 1.

The following comments describe selection criteria for the symbols and their values:

POSIX2\_C\_BIND  
 POSIX2\_C\_DEV  
 POSIX2\_FORT\_DEV  
 POSIX2\_FORT\_RUN  
 POSIX2\_SW\_DEV  
 POSIX2\_UPE

It is possible for some (usually privileged) operations to remove utilities that support these options or otherwise to render these options unsupported. The header files, the *sysconf()* function, or the *getconf* utility will not necessarily detect such actions, in which case they should not be considered as rendering the implementation non-conforming. A test suite should not attempt tests such as:

```
rm /usr/bin/c99
getconf POSIX2_C_DEV
```

POSIX2\_LOCALEDEF

This symbol was introduced to allow implementations to restrict supported locales to only those supplied by the implementation.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/2 is applied, deleting the entry for {POSIX2\_VERSION} since it is not a utility limit minimum value.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/3 is applied, changing the text in Utility Limits from: “utility (see *getconf*) through the *sysconf()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001. The literal names shown in Table 1-3 apply only to the *getconf* utility; the high-level language binding describes the exact form of each name to be used by the interfaces in that binding.” to: “utility (see *getconf*).”.



9303 **C.1.10 Grammar Conventions**

9304 There is no additional rationale provided for this section.

9305 **C.1.11 Utility Description Defaults**9306 This section is arranged with headings in the same order as all the utility descriptions. It is a  
9307 collection of related and unrelated information concerning:

9308 1. The default actions of utilities

9309 2. The meanings of notations used in IEEE Std 1003.1-2001 that are specific to individual  
9310 utility sections9311 Although this material may seem out of place here, it is important that this information appear  
9312 before any of the utilities to be described later.9313 **NAME**

9314 There is no additional rationale provided for this section.

9315 **SYNOPSIS**

9316 There is no additional rationale provided for this section.

9317 **DESCRIPTION**

9318 There is no additional rationale provided for this section.

9319 **OPTIONS**9320 Although it has not always been possible, the standard developers tried to avoid repeating  
9321 information to reduce the risk that duplicate explanations could each be modified differently.9322 The need to recognize `--` is required because conforming applications need to shield their  
9323 operands from any arbitrary options that the implementation may provide as an extension. For  
9324 example, if the standard utility *foo* is listed as taking no options, and the application needed to  
9325 give it a pathname with a leading hyphen, it could safely do it as:9326 `foo -- -myfile`9327 and avoid any problems with `-m` used as an extension.9328 **OPERANDS**9329 The usage of `-` is never shown in the SYNOPSIS. Similarly, the usage of `--` is never shown.9330 The requirement for processing operands in command-line order is to avoid a “WeirdNIX”  
9331 utility that might choose to sort the input files alphabetically, by size, or by directory order.  
9332 Although this might be acceptable for some utilities, in general the programmer has a right to  
9333 know exactly what order will be chosen.9334 Some of the standard utilities take multiple *file* operands and act as if they were processing the  
9335 concatenation of those files. For example:9336 `asa file1 file2`

9337 and:

9338 `cat file1 file2 | asa`

have similar results when questions of file access, errors, and performance are ignored. Other utilities such as *grep* or *wc* have completely different results in these two cases. This latter type of utility is always identified in its DESCRIPTION or OPERANDS sections, whereas the former is not. Although it might be possible to create a general assertion about the former case, the following points must be addressed:

- Access times for the files might be different in the operand case *versus* the *cat* case.
- The utility may have error messages that are cognizant of the input filename, and this added value should not be suppressed. (As an example, *awk* sets a variable with the filename at each file boundary.)

## STDIN

There is no additional rationale provided for this section.

## INPUT FILES

A conforming application cannot assume the following three commands are equivalent:

```
tail -n +2 file
(sed -n 1q; cat) < file
cat file | (sed -n 1q; cat)
```

The second command is equivalent to the first only when the file is seekable. In the third command, if the file offset in the open file description were not unspecified, *sed* would have to be implemented so that it read from the pipe 1 byte at a time or it would have to employ some method to seek backwards on the pipe. Such functionality is not defined currently in POSIX.1 and does not exist on all historical systems. Other utilities, such as *head*, *read*, and *sh*, have similar properties, so the restriction is described globally in this section.

The definition of “text file” is strictly enforced for input to the standard utilities; very few of them list exceptions to the undefined results called for here. (Of course, “undefined” here does not mean that historical implementations necessarily have to change to start indicating error conditions. Conforming applications cannot rely on implementations succeeding or failing when non-text files are used.)

The utilities that allow line continuation are generally those that accept input languages, rather than pure data. It would be unusual for an input line of this type to exceed {LINE\_MAX} bytes and unreasonable to require that the implementation allow unlimited accumulation of multiple lines, each of which could reach {LINE\_MAX}. Thus, for a conforming application the total of all the continued lines in a set cannot exceed {LINE\_MAX}.

The format description is intended to be sufficiently rigorous to allow other applications to generate these input files. However, since <blank>s can legitimately be included in some of the fields described by the standard utilities, particularly in locales other than the POSIX locale, this intent is not always realized.

## ENVIRONMENT VARIABLES

There is no additional rationale provided for this section.

**ASYNCHRONOUS EVENTS**

Because there is no language prohibiting it, a utility is permitted to catch a signal, perform some additional processing (such as deleting temporary files), restore the default signal action (or action inherited from the parent process), and resignal itself.

**STDOUT**

The format description is intended to be sufficiently rigorous to allow post-processing of output by other programs, particularly by an *awk* or *lex* parser.

**STDERR**

This section does not describe error messages that refer to incorrect operation of the utility. Consider a utility that processes program source code as its input. This section is used to describe messages produced by a correctly operating utility that encounters an error in the program source code on which it is processing. However, a message indicating that the utility had insufficient memory in which to operate would not be described.

Some utilities have traditionally produced warning messages without returning a non-zero exit status; these are specifically noted in their sections. Other utilities shall not write to standard error if they complete successfully, unless the implementation provides some sort of extension to increase the verbosity or debugging level.

The format descriptions are intended to be sufficiently rigorous to allow post-processing of output by other programs.

**OUTPUT FILES**

The format description is intended to be sufficiently rigorous to allow post-processing of output by other programs, particularly by an *awk* or *lex* parser.

Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging mode) that would bypass any attempted recovery actions.

**EXTENDED DESCRIPTION**

There is no additional rationale provided for this section.

**EXIT STATUS**

Note the additional discussion of exit values in *Exit Status for Commands* in the *sh* utility. It describes requirements for returning exit values greater than 125.

A utility may list zero as a successful return, 1 as a failure for a specific reason, and greater than 1 as “an error occurred”. In this case, unspecified conditions may cause a 2 or 3, or other value, to be returned. A strictly conforming application should be written so that it tests for successful exit status values (zero in this case), rather than relying upon the single specific error value listed in IEEE Std 1003.1-2001. In that way, it will have maximum portability, even on implementations with extensions.

The standard developers are aware that the general non-enumeration of errors makes it difficult to write test suites that test the *incorrect* operation of utilities. There are some historical implementations that have expended effort to provide detailed status messages and a helpful environment to bypass or explain errors, such as prompting, retrying, or ignoring unimportant syntax errors; other implementations have not. Since there is no realistic way to mandate system behavior in cases of undefined application actions or system problems—in a manner acceptable to all cultures and environments—attention has been limited to the correct operation of utilities

9419 by the conforming application. Furthermore, the conforming application does not need detailed  
 9420 information concerning errors that it caused through incorrect usage or that it cannot correct.

9421 There is no description of defaults for this section because all of the standard utilities specify  
 9422 something (or explicitly state “Unspecified”) for exit status.

#### 9423 **CONSEQUENCES OF ERRORS**

9424 Several actions are possible when a utility encounters an error condition, depending on the  
 9425 severity of the error and the state of the utility. Included in the possible actions of various  
 9426 utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; and  
 9427 validity checking of the file system or directory.

9428 The text about recursive traversing is meant to ensure that utilities such as *find* process as many  
 9429 files in the hierarchy as they can. They should not abandon all of the hierarchy at the first error  
 9430 and resume with the next command-line operand, but should attempt to keep going.

#### 9431 **APPLICATION USAGE**

9432 This section provides additional caveats, issues, and recommendations to the developer.

#### 9433 **EXAMPLES**

9434 This section provides sample usage.

#### 9435 **RATIONALE**

9436 There is no additional rationale provided for this section.

#### 9437 **FUTURE DIRECTIONS**

9438 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
 9439 the future, and often cautions the developer to architect the code to account for a change in this  
 9440 area. Note that a future directions statement should not be taken as a commitment to adopt a  
 9441 feature or interface in the future.

#### 9442 **SEE ALSO**

9443 There is no additional rationale provided for this section.

#### 9444 **CHANGE HISTORY**

9445 There is no additional rationale provided for this section.

### 9446 **C.1.12 Considerations for Utilities in Support of Files of Arbitrary Size**

9447 This section is intended to clarify the requirements for utilities in support of large files.

9448 The utilities listed in this section are utilities which are used to perform administrative tasks  
 9449 such as to create, move, copy, remove, change the permissions, or measure the resources of a  
 9450 file. They are useful both as end-user tools and as utilities invoked by applications during  
 9451 software installation and operation.

9452 The *chgrp*, *chmod*, *chown*, *ln*, and *rm* utilities probably require use of large file-capable versions of  
 9453 *stat()*, *lstat()*, *ftw()*, and the **stat** structure.

9454 The *cat*, *cksum*, *cmp*, *cp*, *dd*, *mv*, *sum*, and *touch* utilities probably require use of large file-capable  
 9455 versions of *creat()*, *open()*, and *fopen()*.

The *cat*, *cksum*, *cmp*, *dd*, *df*, *du*, *ls*, and *sum* utilities may require writing large integer values. For example:

- The *cat* utility might have a **-n** option which counts <newline>s.
- The *cksum* and *ls* utilities report file sizes.
- The *cmp* utility reports the line number at which the first difference occurs, and also has a **-l** option which reports file offsets.
- The *dd*, *df*, *du*, *ls*, and *sum* utilities report block counts.

The *dd*, *find*, and *test* utilities may need to interpret command arguments that contain 64-bit values. For *dd*, the arguments include *skip=n*, *seek=n*, and *count=n*. For *find*, the arguments include **-sizen**. For *test*, the arguments are those associated with algebraic comparisons.

The *df* utility might need to access large file systems with *statvfs()*.

The *ulimit* utility will need to use large file-capable versions of *getrlimit()* and *setrlimit()* and be able to read and write large integer values.

### C.1.13 Built-In Utilities

All of these utilities can be *exec*-ed. There is no requirement that these utilities are actually built into the shell itself, but many shells need the capability to do so because the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.9.1.1, Command Search and Execution requires that they be found prior to the *PATH* search. The shell could satisfy its requirements by keeping a list of the names and directly accessing the file-system versions regardless of *PATH*. Providing all of the required functionality for those such as *cd* or *read* would be more difficult.

There were originally three justifications for allowing the omission of *exec*-able versions:

1. It would require wasting space in the file system, at the expense of very small systems. However, it has been pointed out that all 16 utilities in the table can be provided with 16 links to a single-line shell script:

```
$0 "$@"
```

2. It is not logical to require invocation of utilities such as *cd* because they have no value outside the shell environment or cannot be useful in a child process. However, counter-examples always seemed to be available for even the most unusual cases:

```
find . -type d -exec cd {} \; -exec foo {} \;
 (which invokes "foo" on accessible directories)
```

```
ps ... | sed ... | xargs kill
```

```
find . -exec true \; -a ...
 (where "true" is used for temporary debugging)
```

3. It is confusing to have a utility such as *kill* that can easily be in the file system in the base standard, but that requires built-in status for the User Portability Utilities option (for the % job control job ID notation). It was decided that it was more appropriate to describe the required functionality (rather than the implementation) to the system implementors and let them decide how to satisfy it.

On the other hand, it was realized that any distinction like this between utilities was not useful to applications, and that the cost to correct it was small. These arguments were ultimately the most effective.

|      |                                                                                                                     |
|------|---------------------------------------------------------------------------------------------------------------------|
| 9497 | There were varying reasons for including utilities in the table of built-ins:                                       |
| 9498 | <i>alias, fc, unalias</i>                                                                                           |
| 9499 | The functionality of these utilities is performed more simply within the shell itself and that                      |
| 9500 | is the model most historical implementations have used.                                                             |
| 9501 | <i>bg, fg, jobs</i>                                                                                                 |
| 9502 | All of the job control-related utilities are eligible for built-in status because that is the model                 |
| 9503 | most historical implementations have used.                                                                          |
| 9504 | <i>cd, getopts, newgrp, read, umask, wait</i>                                                                       |
| 9505 | The functionality of these utilities is performed more simply within the context of the                             |
| 9506 | current process. An example can be taken from the usage of the <i>cd</i> utility. The purpose of                    |
| 9507 | the <i>cd</i> utility is to change the working directory for subsequent operations. The actions of <i>cd</i>        |
| 9508 | affect the process in which <i>cd</i> is executed and all subsequent child processes of that process.               |
| 9509 | Based on the POSIX standard process model, changes in the process environment of a child                            |
| 9510 | process have no effect on the parent process. If the <i>cd</i> utility were executed from a child                   |
| 9511 | process, the working directory change would be effective only in the child process. Child                           |
| 9512 | processes initiated subsequent to the child process that executed the <i>cd</i> utility would not                   |
| 9513 | have a changed working directory relative to the parent process.                                                    |
| 9514 | <i>command</i>                                                                                                      |
| 9515 | This utility was placed in the table primarily to protect scripts that are concerned about                          |
| 9516 | their <i>PATH</i> being manipulated. The “secure” shell script example in the <i>command</i> utility in             |
| 9517 | the Shell and Utilities volume of IEEE Std 1003.1-2001 would not be possible if a <i>PATH</i>                       |
| 9518 | change retrieved an alien version of <i>command</i> . (An alternative would have been to                            |
| 9519 | implement <i>getconf</i> as a built-in, but the standard developers considered that it carried too                  |
| 9520 | many changing configuration strings to require in the shell.)                                                       |
| 9521 | <i>kill</i> Since <i>kill</i> provides optional job control functionality using shell notation (%1, %2, and so on), |
| 9522 | some implementations would find it extremely difficult to provide this outside the shell.                           |
| 9523 | <i>true, false</i>                                                                                                  |
| 9524 | These are in the table as a courtesy to programmers who wish to use the "while true"                                |
| 9525 | shell construct without protecting <i>true</i> from <i>PATH</i> searches. (It is acknowledged that                  |
| 9526 | "while :" also works, but the idiom with <i>true</i> is historically pervasive.)                                    |
| 9527 | All utilities, including those in the table, are accessible via the <i>system()</i> and <i>popen()</i> functions in |
| 9528 | the System Interfaces volume of IEEE Std 1003.1-2001. There are situations where the return                         |
| 9529 | functionality of <i>system()</i> and <i>popen()</i> is not desirable. Applications that require the exit status of  |
| 9530 | the invoked utility will not be able to use <i>system()</i> or <i>popen()</i> , since the exit status returned is   |
| 9531 | that of the command language interpreter rather than that of the invoked utility. The alternative                   |
| 9532 | for such applications is the use of the <i>exec</i> family.                                                         |

## 9533 C.2 Shell Command Language

### 9534 C.2.1 Shell Introduction

9535 The System V shell was selected as the starting point for the Shell and Utilities volume of  
 9536 IEEE Std 1003.1-2001. The BSD C shell was excluded from consideration for the following  
 9537 reasons:

- 9538 • Most historically portable shell scripts assume the Version 7 Bourne shell, from which the  
 9539 System V shell is derived.
- 9540 • The majority of tutorial materials on shell programming assume the System V shell.

9541 The construct "#!" is reserved for implementations wishing to provide that extension. If it were  
 9542 not reserved, the Shell and Utilities volume of IEEE Std 1003.1-2001 would disallow it by forcing  
 9543 it to be a comment. As it stands, a strictly conforming application must not use "#!" as the first  
 9544 two characters of the file.

### 9545 C.2.2 Quoting

9546 There is no additional rationale provided for this section.

#### 9547 C.2.2.1 Escape Character (Backslash)

9548 There is no additional rationale provided for this section.

#### 9549 C.2.2.2 Single-Quotes

9550 A backslash cannot be used to escape a single-quote in a single-quoted string. An embedded  
 9551 quote can be created by writing, for example: "'a'\''b'", which yields "a'b". (See the Shell  
 9552 and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.5, Field Splitting for a better  
 9553 understanding of how portions of words are either split into fields or remain concatenated.) A  
 9554 single token can be made up of concatenated partial strings containing all three kinds of quoting  
 9555 or escaping, thus permitting any combination of characters.

#### 9556 C.2.2.3 Double-Quotes

9557 The escaped <newline> used for line continuation is removed entirely from the input and is not  
 9558 replaced by any white space. Therefore, it cannot serve as a token separator.

9559 In double-quoting, if a backslash is immediately followed by a character that would be  
 9560 interpreted as having a special meaning, the backslash is deleted and the subsequent character is  
 9561 taken literally. If a backslash does not precede a character that would have a special meaning, it  
 9562 is left in place unmodified and the character immediately following it is also left unmodified.  
 9563 Thus, for example:

9564       "\\$" → \$

9565       "a" → \a

9566 It would be desirable to include the statement “The characters from an enclosed "\${" to the  
 9567 matching '}' shall not be affected by the double quotes”, similar to the one for "\$()".  
 9568 However, historical practice in the System V shell prevents this.

9569 The requirement that double-quotes be matched inside "\${...}" within double-quotes and the  
 9570 rule for finding the matching '}' in the Shell and Utilities volume of IEEE Std 1003.1-2001,  
 9571 Section 2.6.2, Parameter Expansion eliminate several subtle inconsistencies in expansion for  
 9572 historical shells in rare cases; for example:

9573           `"${foo-bar}"`

9574           yields **bar** when **foo** is not defined, and is an invalid substitution when **foo** is defined, in many  
 9575           historical shells. The differences in processing the `"${...}"` form have led to inconsistencies  
 9576           between historical systems. A consequence of this rule is that single-quotes cannot be used to  
 9577           quote the `' '` within `"${...}"`; for example:

9578           `unset bar`  
 9579           `foo="${bar-' }'"`

9580           is invalid because the `"${...}"` substitution contains an unpaired unescaped single-quote. The  
 9581           backslash can be used to escape the `' '` in this example to achieve the desired result:

9582           `unset bar`  
 9583           `foo="${bar-\ }"`

9584           The differences in processing the `"${...}"` form have led to inconsistencies between the  
 9585           historical System V shell, BSD, and KornShells, and the text in the Shell and Utilities volume of  
 9586           IEEE Std 1003.1-2001 is an attempt to converge them without breaking too many applications.  
 9587           The only alternative to this compromise between shells would be to make the behavior  
 9588           unspecified whenever the literal characters `' '`, `'{'`, `' '`, and `'"'` appear within `"${...}"`.  
 9589           To write a portable script that uses these values, a user would have to assign variables; for  
 9590           example:

9591           `squote=\' dquote=\" lbrace='{ ' rbrace='}'`  
 9592           `${foo-$squote$rbrace$squote}`

9593           rather than:

9594           `${foo-"'"'"' }`

9595           Some implementations have allowed the end of the word to terminate the backquoted command  
 9596           substitution, such as in:

9597           `"`echo hello`"`

9598           This usage is undefined; the matching backquote is required by the Shell and Utilities volume of  
 9599           IEEE Std 1003.1-2001. The other undefined usage can be illustrated by the example:

9600           `sh -c '` echo "foo`'`

9601           The description of the recursive actions involving command substitution can be illustrated with  
 9602           an example. Upon recognizing the introduction of command substitution, the shell parses input  
 9603           (in a new context), gathering the source for the command substitution until an unbalanced `' ) '`  
 9604           or `' '`` is located. For example, in the following:

9605           `echo "$(date; echo "`  
 9606           `one" )"`

9607           the double-quote following the *echo* does not terminate the first double-quote; it is part of the  
 9608           command substitution script. Similarly, in:

9609           `echo "$(echo *)"`

9610           the asterisk is not quoted since it is inside command substitution; however:

9611           `echo "$(echo "*)"`

9612           is quoted (and represents the asterisk character itself).



### 9613 C.2.3 Token Recognition

9614 The " ( ( " and " ) ) " symbols are control operators in the KornShell, used for an alternative  
 9615 syntax of an arithmetic expression command. A conforming application cannot use " ( ( " as a  
 9616 single token (with the exception of the "\$ ( ( " form for shell arithmetic).

9617 On some implementations, the symbol " ( ( " is a control operator; its use produces unspecified  
 9618 results. Applications that wish to have nested subshells, such as:

```
9619 ((echo Hello);(echo World))
```

9620 must separate the " ( ( " characters into two tokens by including white space between them.  
 9621 Some systems may treat these as invalid arithmetic expressions instead of subshells.

9622 Certain combinations of characters are invalid in portable scripts, as shown in the grammar.  
 9623 Implementations may use these combinations (such as " | & ") as valid control operators. Portable  
 9624 scripts cannot rely on receiving errors in all cases where this volume of IEEE Std 1003.1-2001  
 9625 indicates that a syntax is invalid.

9626 The (3) rule about combining characters to form operators is not meant to preclude systems from  
 9627 extending the shell language when characters are combined in otherwise invalid ways.  
 9628 Conforming applications cannot use invalid combinations, and test suites should not penalize  
 9629 systems that take advantage of this fact. For example, the unquoted combination " | & " is not  
 9630 valid in a POSIX script, but has a specific KornShell meaning.

9631 The (10) rule about ' #' as the current character is the first in the sequence in which a new token  
 9632 is being assembled. The ' #' starts a comment only when it is at the beginning of a token. This  
 9633 rule is also written to indicate that the search for the end-of-comment does not consider escaped  
 9634 <newline> specially, so that a comment cannot be continued to the next line.

#### 9635 C.2.3.1 Alias Substitution

9636 The alias capability was added in the User Portability Utilities option because it is widely used in  
 9637 historical implementations by interactive users.

9638 The definition of "alias name" precludes an alias name containing a slash character. Since the  
 9639 text applies to the command words of simple commands, reserved words (in their proper  
 9640 places) cannot be confused with aliases.

9641 The placement of alias substitution in token recognition makes it clear that it precedes all of the  
 9642 word expansion steps.

9643 An example concerning trailing <blank>s and reserved words follows. If the user types:

```
9644 $ alias foo="/bin/ls "

 9645 $ alias while="/"
```

9646 The effect of executing:

```
9647 $ while true

 9648 > do

 9649 > echo "Hello, World"

 9650 > done
```

9651 is a never-ending sequence of "Hello, World" strings to the screen. However, if the user  
 9652 types:

```
9653 $ foo while
```

9654 the result is an *ls* listing of /. Since the alias substitution for **foo** ends in a <space>, the next word  
 9655 is checked for alias substitution. The next word, **while**, has also been aliased, so it is substituted

9656 as well. Since it is not in the proper position as a command word, it is not recognized as a  
 9657 reserved word.

9658 If the user types:

9659 `$ foo; while`

9660 **while** retains its normal reserved-word properties.

#### 9661 C.2.4 Reserved Words

9662 All reserved words are recognized syntactically as such in the contexts described. However, note  
 9663 that **in** is the only meaningful reserved word after a **case** or **for**; similarly, **in** is not meaningful as  
 9664 the first word of a simple command.

9665 Reserved words are recognized only when they are delimited (that is, meet the definition of the  
 9666 Base Definitions volume of IEEE Std 1003.1-2001, Section 3.435, Word), whereas operators are  
 9667 themselves delimiters. For instance, ' ( ' and ' ) ' are control operators, so that no <space> is  
 9668 needed in (*list*). However, ' { ' and ' } ' are reserved words in { *list*; }, so that in this case the  
 9669 leading <space> and semicolon are required.

9670 The list of unspecified reserved words is from the KornShell, so conforming applications cannot  
 9671 use them in places a reserved word would be recognized. This list contained **time** in early  
 9672 proposals, but it was removed when the *time* utility was selected for the Shell and Utilities  
 9673 volume of IEEE Std 1003.1-2001.

9674 There was a strong argument for promoting braces to operators (instead of reserved words), so  
 9675 they would be syntactically equivalent to subshell operators. Concerns about compatibility  
 9676 outweighed the advantages of this approach. Nevertheless, conforming applications should  
 9677 consider quoting ' { ' and ' } ' when they represent themselves.

9678 The restriction on ending a name with a colon is to allow future implementations that support  
 9679 named labels for flow control; see the RATIONALE for the *break* built-in utility.

9680 It is possible that a future version of the Shell and Utilities volume of IEEE Std 1003.1-2001 may  
 9681 require that ' { ' and ' } ' be treated individually as control operators, although the token " { } "  
 9682 will probably be a special-case exemption from this because of the often-used *find*{ } construct.

#### 9683 C.2.5 Parameters and Variables

##### 9684 C.2.5.1 Positional Parameters

9685 There is no additional rationale provided for this section.

##### 9686 C.2.5.2 Special Parameters

9687 Most historical implementations implement subshells by forking; thus, the special parameter  
 9688 ' \$ ' does not necessarily represent the process ID of the shell process executing the commands  
 9689 since the subshell execution environment preserves the value of ' \$ '.

9690 If a subshell were to execute a background command, the value of " \$ ! " for the parent would  
 9691 not change. For example:

```
9692 (
9693 date &
9694 echo $!
9695)
9696 echo $!
```

9697 would echo two different values for "\$!".

9698 The "\$-" special parameter can be used to save and restore *set* options:

```
9699 Save=$(echo $- | sed 's/[ics]//g')
9700 ...
9701 set +aCefnuvx
9702 if [-n "$Save"]; then
9703 set -$Save
9704 fi
```

9705 The three options are removed using *sed* in the example because they may appear in the value of  
9706 "\$-" (from the *sh* command line), but are not valid options to *set*.

9707 The descriptions of parameters '\*' and '@' assume the reader is familiar with the field splitting  
9708 discussion in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.5, Field Splitting  
9709 and understands that portions of the word remain concatenated unless there is some reason to  
9710 split them into separate fields.

9711 Some examples of the '\*' and '@' properties, including the concatenation aspects:

```
9712 set "abc" "def ghi" "jkl"
9713 echo $* => "abc" "def" "ghi" "jkl"
9714 echo "$*" => "abc def ghi jkl"
9715 echo $@ => "abc" "def" "ghi" "jkl"
```

9716 but:

```
9717 echo "$@" => "abc" "def ghi" "jkl"
9718 echo "xx$@yy" => "xxabc" "def ghi" "jkl"
9719 echo "$@$@" => "abc" "def ghi" "jklabc" "def ghi" "jkl"
```

9720 In the preceding examples, the double-quote characters that appear after the "=>" do not appear  
9721 in the output and are used only to illustrate word boundaries.

9722 The following example illustrates the effect of setting *IFS* to a null string:

```
9723 $ IFS=''
9724 $ set foo bar bam
9725 $ echo "$@"
9726 foo bar bam
9727 $ echo "$*"
9728 foobarbam
9729 $ unset IFS
9730 $ echo "$*"
9731 foo bar bam
```

### 9732 C.2.5.3 Shell Variables

9733 See the discussion of *IFS* in Section C.2.6.5 (on page 241) and the RATIONALE for the *sh* utility.

9734 The prohibition on *LC\_CTYPE* changes affecting lexical processing protects the shell  
9735 implementor (and the shell programmer) from the ill effects of changing the definition of  
9736 <blank> or the set of alphabetic characters in the current environment. It would probably not be  
9737 feasible to write a compiled version of a shell script without this rule. The rule applies only to  
9738 the current invocation of the shell and its subshells—invoking a shell script or performing *exec sh*  
9739 would subject the new shell to the changes in *LC\_CTYPE*.

|      |                |                                                                                                   |
|------|----------------|---------------------------------------------------------------------------------------------------|
| 9740 |                | Other common environment variables used by historical shells are not specified by the Shell and   |
| 9741 |                | Utilities volume of IEEE Std 1003.1-2001, but they should be reserved for the historical uses.    |
| 9742 |                | Tilde expansion for components of <i>PATH</i> in an assignment such as:                           |
| 9743 |                | <code>PATH=~hlj/bin:~dwc/bin:\$PATH</code>                                                        |
| 9744 |                | is a feature of some historical shells and is allowed by the wording of the Shell and Utilities   |
| 9745 |                | volume of IEEE Std 1003.1-2001, Section 2.6.1, Tilde Expansion. Note that the tildes are expanded |
| 9746 |                | during the assignment to <i>PATH</i> , not when <i>PATH</i> is accessed during command search.    |
| 9747 |                | The following entries represent additional information about variables included in the Shell and  |
| 9748 |                | Utilities volume of IEEE Std 1003.1-2001, or rationale for common variables in use by shells that |
| 9749 |                | have been excluded:                                                                               |
| 9750 | —              | (Underscore.) While underscore is historical practice, its overloaded usage in                    |
| 9751 |                | the KornShell is confusing, and it has been omitted from the Shell and Utilities                  |
| 9752 |                | volume of IEEE Std 1003.1-2001.                                                                   |
| 9753 | <i>ENV</i>     | This variable can be used to set aliases and other items local to the invocation                  |
| 9754 |                | of a shell. The file referred to by <i>ENV</i> differs from <b>\$HOME/.profile</b> in that        |
| 9755 |                | <b>.profile</b> is typically executed at session start-up, whereas the <i>ENV</i> file is         |
| 9756 |                | executed at the beginning of each shell invocation. The <i>ENV</i> value is                       |
| 9757 |                | interpreted in a manner similar to a dot script, in that the commands are                         |
| 9758 |                | executed in the current environment and the file needs to be readable, but not                    |
| 9759 |                | executable. However, unlike dot scripts, no <i>PATH</i> searching is performed.                   |
| 9760 |                | This is used as a guard against Trojan Horse security breaches.                                   |
| 9761 | <i>ERRNO</i>   | This variable was omitted from the Shell and Utilities volume of                                  |
| 9762 |                | IEEE Std 1003.1-2001 because the values of error numbers are not defined in                       |
| 9763 |                | IEEE Std 1003.1-2001 in a portable manner.                                                        |
| 9764 | <i>FCEDIT</i>  | Since this variable affects only the <i>fc</i> utility, it has been omitted from this more        |
| 9765 |                | global place. The value of <i>FCEDIT</i> does not affect the command-line editing                 |
| 9766 |                | mode in the shell; see the description of <i>set -o vi</i> in the <i>set</i> built-in utility.    |
| 9767 | <i>PS1</i>     | This variable is used for interactive prompts. Historically, the “superuser”                      |
| 9768 |                | has had a prompt of ‘#’. Since privileges are not required to be monolithic, it                   |
| 9769 |                | is difficult to define which privileges should cause the alternate prompt.                        |
| 9770 |                | However, a sufficiently powerful user should be reminded of that power by                         |
| 9771 |                | having an alternate prompt.                                                                       |
| 9772 | <i>PS3</i>     | This variable is used by the KornShell for the <i>select</i> command. Since the POSIX             |
| 9773 |                | shell does not include <i>select</i> , <i>PS3</i> was omitted.                                    |
| 9774 | <i>PS4</i>     | This variable is used for shell debugging. For example, the following script:                     |
| 9775 |                | <code>PS4='[\${LINENO}] + '</code>                                                                |
| 9776 |                | <code>set -x</code>                                                                               |
| 9777 |                | <code>echo Hello</code>                                                                           |
| 9778 |                | writes the following to standard error:                                                           |
| 9779 |                | <code>[3]+ echo Hello</code>                                                                      |
| 9780 | <i>RANDOM</i>  | This pseudo-random number generator was not seen as being useful to                               |
| 9781 |                | interactive users.                                                                                |
| 9782 | <i>SECONDS</i> | Although this variable is sometimes used with <i>PS1</i> to allow the display of the              |
| 9783 |                | current time in the prompt of the user, it is not one that would be manipulated                   |

9784 frequently enough by an interactive user to include in the Shell and Utilities  
 9785 volume of IEEE Std 1003.1-2001.

## 9786 C.2.6 Word Expansions

9787 Step (2) refers to the “portions of fields generated by step (1)”. For example, if the word being  
 9788 expanded were "\$x+\$y" and *IFS*=+, the word would be split only if "\$x" or "\$y" contained  
 9789 '+'; the '+' in the original word was not generated by step (1).

9790 *IFS* is used for performing field splitting on the results of parameter and command substitution;  
 9791 it is not used for splitting all fields. Previous versions of the shell used it for splitting all fields  
 9792 during field splitting, but this has severe problems because the shell can no longer parse its own  
 9793 script. There are also important security implications caused by this behavior. All useful  
 9794 applications of *IFS* use it for parsing input of the *read* utility and for splitting the results of  
 9795 parameter and command substitution.

9796 The rule concerning expansion to a single field requires that if **foo=abc** and **bar=def**, that:

9797 "\$foo"\$bar"

9798 expands to the single field:

9799 abcdef

9800 The rule concerning empty fields can be illustrated by:

```
9801 $ unset foo
9802 $ set $foo bar ' ' xyz "$foo" abc
9803 $ for i
9804 > do
9805 > echo "-$i-"
9806 > done
9807 -bar-
9808 --
9809 -xyz-
9810 --
9811 -abc-
```

9812 Step (1) indicates that parameter expansion, command substitution, and arithmetic expansion  
 9813 are all processed simultaneously as they are scanned. For example, the following is valid  
 9814 arithmetic:

```
9815 x=1
9816 echo $(($(echo 3)+$x))
```

9817 An early proposal stated that tilde expansion preceded the other steps, but this is not the case in  
 9818 known historical implementations; if it were, and if a referenced home directory contained a '\$'  
 9819 character, expansions would result within the directory name.

### 9820 C.2.6.1 Tilde Expansion

9821 Tilde expansion generally occurs only at the beginning of words, but an exception based on  
 9822 historical practice has been included:

```
9823 PATH=/posix/bin:~dgg/bin
```

9824 This is eligible for tilde expansion because tilde follows a colon and none of the relevant  
 9825 characters is quoted. Consideration was given to prohibiting this behavior because any of the  
 9826 following are reasonable substitutes:

```

9827 PATH=$(printf %s ~karels/bin : ~bostic/bin)
9828 for Dir in ~maat/bin ~srb/bin ...
9829 do
9830 PATH=${PATH:+$PATH:}$Dir
9831 done

```

9832 In the first command, explicit colons are used for each directory. In all cases, the shell performs  
 9833 tilde expansion on each directory because all are separate words to the shell.

9834 Note that expressions in operands such as:

```

9835 make -k mumble LIBDIR=~chet/lib

```

9836 do not qualify as shell variable assignments, and tilde expansion is not performed (unless the  
 9837 command does so itself, which *make* does not).

9838 Because of the requirement that the word is not quoted, the following are not equivalent; only  
 9839 the last causes tilde expansion:

```

9840 \~hlj/ ~h\lj/ ~"hlj"/ ~hlj\ / ~hlj/

```

9841 In an early proposal, tilde expansion occurred following any unquoted equals sign or colon, but  
 9842 this was removed because of its complexity and to avoid breaking commands such as:

```

9843 rcp hostname:~marc/.profile .

```

9844 A suggestion was made that the special sequence "\$~" should be allowed to force tilde  
 9845 expansion anywhere. Since this is not historical practice, it has been left for future  
 9846 implementations to evaluate. (The description in the Shell and Utilities volume of  
 9847 IEEE Std 1003.1-2001, Section 2.2, Quoting requires that a dollar sign be quoted to represent  
 9848 itself, so the "\$~" combination is already unspecified.)

9849 The results of giving tilde with an unknown login name are undefined because the KornShell  
 9850 "~+" and "~-" constructs make use of this condition, but in general it is an error to give an  
 9851 incorrect login name with tilde. The results of having *HOME* unset are unspecified because some  
 9852 historical shells treat this as an error.

#### 9853 C.2.6.2 Parameter Expansion

9854 The rule for finding the closing '}' in "\${...}" is the one used in the KornShell and is  
 9855 upwardly-compatible with the Bourne shell, which does not determine the closing '}' until the  
 9856 word is expanded. The advantage of this is that incomplete expansions, such as:

```

9857 ${foo

```

9858 can be determined during tokenization, rather than during expansion.

9859 The string length and substring capabilities were included because of the demonstrated need for  
 9860 them, based on their usage in other shells, such as C shell and KornShell.

9861 Historical versions of the KornShell have not performed tilde expansion on the word part of  
 9862 parameter expansion; however, it is more consistent to do so.

## 9863 C.2.6.3 Command Substitution

9864 The "\$ ( )" form of command substitution solves a problem of inconsistent behavior when using  
 9865 backquotes. For example:

| 9866 | Command             | Output |
|------|---------------------|--------|
| 9867 | echo '\$x'          | \$x    |
| 9868 | echo `echo '\$x'`   | \$x    |
| 9869 | echo \$(echo '\$x') | \$x    |

9870 Additionally, the backquoted syntax has historical restrictions on the contents of the embedded  
 9871 command. While the newer "\$ ( )" form can process any kind of valid embedded script, the  
 9872 backquoted form cannot handle some valid scripts that include backquotes. For example, these  
 9873 otherwise valid embedded scripts do not work in the left column, but do work on the right:

|      |                             |                             |
|------|-----------------------------|-----------------------------|
| 9874 | echo `                      | echo \$(                    |
| 9875 | cat <<\eof                  | cat <<\eof                  |
| 9876 | a here-doc with `           | a here-doc with )           |
| 9877 | eof                         | eof                         |
| 9878 | `                           | )                           |
| 9879 | echo `                      | echo \$(                    |
| 9880 | echo abc # a comment with ` | echo abc # a comment with ) |
| 9881 | `                           | )                           |
| 9882 | echo `                      | echo \$(                    |
| 9883 | echo ` `                    | echo ` `                    |
| 9884 | `                           | )                           |

9885 Because of these inconsistent behaviors, the backquoted variety of command substitution is not  
 9886 recommended for new applications that nest command substitutions or attempt to embed  
 9887 complex scripts.

9888 The KornShell feature:

9889 If *command* is of the form *<word*, *word* is expanded to generate a pathname, and the value of  
 9890 the command substitution is the contents of this file with any trailing *<newline>*s deleted.

9891 was omitted from the Shell and Utilities volume of IEEE Std 1003.1-2001 because \$(*cat word*) is  
 9892 an appropriate substitute. However, to prevent breaking numerous scripts relying on this  
 9893 feature, it is unspecified to have a script within "\$ ( )" that has only redirections.

9894 The requirement to separate "\$ (" and ' (' when a single subshell is command-substituted is to  
 9895 avoid any ambiguities with arithmetic expansion.

9896 IEEE Std 1003.1-2001 / Cor 1-2002, item XCU / TC1 / D6 / 4 is applied, changing the text from: "If a  
 9897 command substitution occurs inside double-quotes, it shall not be performed on the results of  
 9898 the substitution." to: "If a command substitution occurs inside double-quotes, field splitting and  
 9899 pathname expansion shall not be performed on the results of the substitution.". The  
 9900 replacement text taken from the ISO POSIX-2:1993 standard is clearer about the items that are  
 9901 not performed.

## 9902 C.2.6.4 Arithmetic Expansion

9903 The "`(( ))`" form of KornShell arithmetic in early proposals was omitted. The standard  
 9904 developers concluded that there was a strong desire for some kind of arithmetic evaluator to  
 9905 replace *expr*, and that relating it to '`$`' makes it work well with the standard shell language, and  
 9906 it provides access to arithmetic evaluation in places where accessing a utility would be  
 9907 inconvenient.

9908 The syntax and semantics for arithmetic were changed for the ISO/IEC 9945-2:1993 standard.  
 9909 The language is essentially a pure arithmetic evaluator of constants and operators (excluding  
 9910 assignment) and represents a simple subset of the previous arithmetic language (which was  
 9911 derived from the KornShell "`(( ))`" construct). The syntax was changed from that of a  
 9912 command denoted by `((expression))` to an expansion denoted by `$(expression)`. The new form is  
 9913 a dollar expansion ('`$`') that evaluates the expression and substitutes the resulting value.  
 9914 Objections to the previous style of arithmetic included that it was too complicated, did not fit in  
 9915 well with the use of variables in the shell, and its syntax conflicted with subshells. The  
 9916 justification for the new syntax is that the shell is traditionally a macro language, and if a new  
 9917 feature is to be added, it should be accomplished by extending the capabilities presented by the  
 9918 current model of the shell, rather than by inventing a new one outside the model; adding a new  
 9919 dollar expansion was perceived to be the most intuitive and least destructive way to add such a  
 9920 new capability.

9921 In early proposals, a form `$(expression)` was used. It was functionally equivalent to the "`$(( ))`"  
 9922 of the current text, but objections were lodged that the 1988 KornShell had already implemented  
 9923 "`$(( ))`" and there was no compelling reason to invent yet another syntax. Furthermore, the  
 9924 "`$[ ]`" syntax had a minor incompatibility involving the patterns in **case** statements.

9925 The portion of the ISO C standard arithmetic operations selected corresponds to the operations  
 9926 historically supported in the KornShell.

9927 It was concluded that the *test* command (**I**) was sufficient for the majority of relational arithmetic  
 9928 tests, and that tests involving complicated relational expressions within the shell are rare, yet  
 9929 could still be accommodated by testing the value of "`$(( ))`" itself. For example:

```
9930 # a complicated relational expression
9931 while [$(((($x + $y)/($a * $b)) < ($foo*$bar))) -ne 0]
```

9932 or better yet, the rare script that has many complex relational expressions could define a  
 9933 function like this:

```
9934 val() {
9935 return $((!$1))
9936 }
```

9937 and complicated tests would be less intimidating:

```
9938 while val $(((($x + $y)/($a * $b)) < ($foo*$bar)))
9939 do
9940 # some calculations
9941 done
```

9942 A suggestion that was not adopted was to modify *true* and *false* to take an optional argument,  
 9943 and *true* would exit true only if the argument was non-zero, and *false* would exit false only if the  
 9944 argument was non-zero:

```
9945 while true $((($x > 5 && $y <= 25))
```

9946 There is a minor portability concern with the new syntax. The example "`$((2+2))`" could have  
 9947 been intended to mean a command substitution of a utility named "`2+2`" in a subshell. The



standard developers considered this to be obscure and isolated to some KornShell scripts (because "\$()" command substitution existed previously only in the KornShell). The text on command substitution requires that the "\$(" and '((' be separate tokens if this usage is needed.

An example such as:

```
echo $((echo hi);(echo there))
```

should not be misinterpreted by the shell as arithmetic because attempts to balance the parentheses pairs would indicate that they are subshells. However, as indicated by the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.112, Control Operator, a conforming application must separate two adjacent parentheses with white space to indicate nested subshells.

Although the ISO/IEC 9899:1999 standard now requires support for **long long** and allows extended integer types with higher ranks, IEEE Std 1003.1-2001 only requires arithmetic expansions to support **signed long** integer arithmetic. Implementations are encouraged to support signed integer values at least as large as the size of the largest file allowed on the implementation.

Implementations are also allowed to perform floating-point evaluations as long as an application won't see different results for expressions that would not overflow **signed long** integer expression evaluation. (This includes appropriate truncation of results to integer values.)

Changes made in response to IEEE PASC Interpretation 1003.2 #208 removed the requirement that the integer constant suffixes `l` and `L` had to be recognized. The ISO POSIX-2:1993 standard did not require the `u`, `ul`, `uL`, `U`, `Ul`, `UL`, `lu`, `lU`, `Lu`, and `LU` suffixes since only signed integer arithmetic was required. Since all arithmetic expressions were treated as handling **signed long** integer types anyway, the `l` and `L` suffixes were redundant. No known scripts used them and some historic shells did not support them. When the ISO/IEC 9899:1999 standard was used as the basis for the description of arithmetic processing, the `ll` and `LL` suffixes and combinations were also not required. Implementations are still free to accept any or all of these suffixes, but are not required to do so.

There was also some confusion as to whether the shell was required to recognize character constants. Syntactically, character constants were required to be recognized, but the requirements for the handling of backslash ('\\') and quote ('''') characters (needed to specify character constants) within an arithmetic expansion were ambiguous. Furthermore, no known shells supported them. Changes made in response to IEEE PASC Interpretation 1003.2 #208 removed the requirement to support them (if they were indeed required before). IEEE Std 1003.1-2001 clearly does not require support for character constants.

#### C.2.6.5 Field Splitting

The operation of field splitting using *IFS*, as described in early proposals, was based on the way the KornShell splits words, but it is incompatible with other common versions of the shell. However, each has merit, and so a decision was made to allow both. If the *IFS* variable is unset or is `<space><tab><newline>`, the operation is equivalent to the way the System V shell splits words. Using characters outside the `<space><tab><newline>` set yields the KornShell behavior, where each of the non-`<space><tab><newline>`s is significant. This behavior, which affords the most flexibility, was taken from the way the original *awk* handled field splitting.

Rule (3) can be summarized as a pseudo-ERE:

```
(s*nS*|s+)
```

where *s* is an *IFS* white space character and *n* is a character in the *IFS* that is not white space. Any string matching that ERE delimits a field, except that the *s+* form does not delimit fields at

9994 the beginning or the end of a line. For example, if *IFS* is <space>/<comma>/<tab>, the string:  
 9995 <space><space>red<space><space>, <space>white<space>blue  
 9996 yields the three colors as the delimited fields.

#### 9997 C.2.6.6 Pathname Expansion

9998 There is no additional rationale provided for this section.

#### 9999 C.2.6.7 Quote Removal

10000 There is no additional rationale provided for this section.

### 10001 C.2.7 Redirection

10002 In the System Interfaces volume of IEEE Std 1003.1-2001, file descriptors are integers in the range  
 10003 0–({OPEN\_MAX}–1). The file descriptors discussed in the Shell and Utilities volume of  
 10004 IEEE Std 1003.1-2001, Section 2.7, Redirection are that same set of small integers.

10005 Having multi-digit file descriptor numbers for I/O redirection can cause some obscure  
 10006 compatibility problems. Specifically, scripts that depend on an example command:

```
10007 echo 22>/dev/null
```

10008 echoing "2" to standard error or "22" to standard output are no longer portable. However, the  
 10009 file descriptor number must still be delimited from the preceding text. For example:

```
10010 cat file2>foo
```

10011 writes the contents of **file2**, not the contents of **file**.

10012 The ">|" format of output redirection was adopted from the KornShell. Along with the  
 10013 *noclobber* option, set **-C**, it provides a safety feature to prevent inadvertent overwriting of  
 10014 existing files. (See the RATIONALE for the *pathchk* utility for why this step was taken.) The  
 10015 restriction on regular files is historical practice.

10016 The System V shell and the KornShell have differed historically on pathname expansion of *word*;  
 10017 the former never performed it, the latter only when the result was a single field (file). As a  
 10018 compromise, it was decided that the KornShell functionality was useful, but only as a shorthand  
 10019 device for interactive users. No reasonable shell script would be written with a command such  
 10020 as:

```
10021 cat foo > a*
```

10022 Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with  
 10023 which they are most comfortable.

10024 The construct "2>&1" is often used to redirect standard error to the same file as standard  
 10025 output. Since the redirections take place beginning to end, the order of redirections is significant.  
 10026 For example:

```
10027 ls > foo 2>&1
```

10028 directs both standard output and standard error to file **foo**. However:

```
10029 ls 2>&1 > foo
```

10030 only directs standard output to file **foo** because standard error was duplicated as standard  
 10031 output before standard output was directed to file **foo**.

10032 The "<>" operator could be useful in writing an application that worked with several terminals,  
 10033 and occasionally wanted to start up a shell. That shell would in turn be unable to run  
 10034 applications that run from an ordinary controlling terminal unless it could make use of "<>"  
 10035 redirection. The specific example is a historical version of the pager *more*, which reads from  
 10036 standard error to get its commands, so standard input and standard output are both available  
 10037 for their usual usage. There is no way of saying the following in the shell without "<>":

```
10038 cat food | more - >/dev/tty03 2<>/dev/tty03
```

10039 Another example of "<>" is one that opens **/dev/tty** on file descriptor 3 for reading and writing:

```
10040 exec 3<> /dev/tty
```

10041 An example of creating a lock file for a critical code region:

```
10042 set -C
10043 until 2> /dev/null > lockfile
10044 do sleep 30
10045 done
10046 set +C
10047 perform critical function
10048 rm lockfile
```

10049 Since **/dev/null** is not a regular file, no error is generated by redirecting to it in *noclobber* mode.

10050 Tilde expansion is not performed on a here-document because the data is treated as if it were  
 10051 enclosed in double quotes.

#### 10052 C.2.7.1 *Redirecting Input*

10053 There is no additional rationale provided for this section.

#### 10054 C.2.7.2 *Redirecting Output*

10055 There is no additional rationale provided for this section.

#### 10056 C.2.7.3 *Appending Redirected Output*

10057 Note that when a file is opened (even with the O\_APPEND flag set), the initial file offset for that  
 10058 file is set to the beginning of the file. Some historic shells set the file offset to the current end-of-  
 10059 file when **append** mode shell redirection was used, but this is not allowed by  
 10060 IEEE Std 1003.1-2001.

#### 10061 C.2.7.4 *Here-Document*

10062 There is no additional rationale provided for this section.

#### 10063 C.2.7.5 *Duplicating an Input File Descriptor*

10064 There is no additional rationale provided for this section.

#### 10065 C.2.7.6 *Duplicating an Output File Descriptor*

10066 There is no additional rationale provided for this section.

10067 C.2.7.7 *Open File Descriptors for Reading and Writing*

10068 There is no additional rationale provided for this section.

10069 **C.2.8 Exit Status and Errors**10070 C.2.8.1 *Consequences of Shell Errors*

10071 There is no additional rationale provided for this section.

10072 C.2.8.2 *Exit Status for Commands*

10073 There is a historical difference in *sh* and *ksh* non-interactive error behavior. When a command  
 10074 named in a script is not found, some implementations of *sh* exit immediately, but *ksh* continues  
 10075 with the next command. Thus, the Shell and Utilities volume of IEEE Std 1003.1-2001 says that  
 10076 the shell “may” exit in this case. This puts a small burden on the programmer, who has to test  
 10077 for successful completion following a command if it is important that the next command not be  
 10078 executed if the previous command was not found. If it is important for the command to have  
 10079 been found, it was probably also important for it to complete successfully. The test for successful  
 10080 completion would not need to change.

10081 Historically, shells have returned an exit status of  $128+n$ , where  $n$  represents the signal number.  
 10082 Since signal numbers are not standardized, there is no portable way to determine which signal  
 10083 caused the termination. Also, it is possible for a command to exit with a status in the same range  
 10084 of numbers that the shell would use to report that the command was terminated by a signal.  
 10085 Implementations are encouraged to choose exit values greater than 256 to indicate programs  
 10086 that terminate by a signal so that the exit status cannot be confused with an exit status generated  
 10087 by a normal termination.

10088 Historical shells make the distinction between “utility not found” and “utility found but cannot  
 10089 execute” in their error messages. By specifying two seldomly used exit status values for these  
 10090 cases, 127 and 126 respectively, this gives an application the opportunity to make use of this  
 10091 distinction without having to parse an error message that would probably change from locale to  
 10092 locale. The *command*, *env*, *nohup*, and *xargs* utilities in the Shell and Utilities volume of  
 10093 IEEE Std 1003.1-2001 have also been specified to use this convention.

10094 When a command fails during word expansion or redirection, most historical implementations  
 10095 exit with a status of 1. However, there was some sentiment that this value should probably be  
 10096 much higher so that an application could distinguish this case from the more normal exit status  
 10097 values. Thus, the language “greater than zero” was selected to allow either method to be  
 10098 implemented.

10099 **C.2.9 Shell Commands**

10100 A description of an “empty command” was removed from an early proposal because it is only  
 10101 relevant in the cases of *sh -c " "*, *system(" ")*, or an empty shell-script file (such as the  
 10102 implementation of *true* on some historical systems). Since it is no longer mentioned in the Shell  
 10103 and Utilities volume of IEEE Std 1003.1-2001, it falls into the silently unspecified category of  
 10104 behavior where implementations can continue to operate as they have historically, but  
 10105 conforming applications do not construct empty commands. (However, note that *sh* does  
 10106 explicitly state an exit status for an empty string or file.) In an interactive session or a script with  
 10107 other commands, extra <newline>s or semicolons, such as:

```

10108 $ false
10109 $
10110 $ echo $?
10111 1

```

10112 would not qualify as the empty command described here because they would be consumed by  
 10113 other parts of the grammar.

#### 10114 C.2.9.1 Simple Commands

10115 The enumerated list is used only when the command is actually going to be executed. For  
 10116 example, in:

```

10117 true || $foo *

```

10118 no expansions are performed.

10119 The following example illustrates both how a variable assignment without a command name  
 10120 affects the current execution environment, and how an assignment with a command name only  
 10121 affects the execution environment of the command:

```

10122 $ x=red
10123 $ echo $x
10124 red
10125 $ export x
10126 $ sh -c 'echo $x'
10127 red
10128 $ x=blue sh -c 'echo $x'
10129 blue
10130 $ echo $x
10131 red

```

10132 This next example illustrates that redirections without a command name are still performed:

```

10133 $ ls foo
10134 ls: foo: no such file or directory
10135 $ > foo
10136 $ ls foo
10137 foo

```

10138 A command without a command name, but one that includes a command substitution, has an  
 10139 exit status of the last command substitution that the shell performed. For example:

```

10140 if x=$(command)
10141 then ...
10142 fi

```

10143 An example of redirections without a command name being performed in a subshell shows that  
 10144 the here-document does not disrupt the standard input of the **while** loop:

```

10145 IFS=:
10146 while read a b
10147 do echo $a
10148 <<-eof
10149 Hello
10150 eof
10151 done </etc/passwd

```

10152       Following are examples of commands without command names in AND-OR lists:

```
10153 > foo || {
10154 echo "error: foo cannot be created" >&2
10155 exit 1
10156 }

10157 # set saved if /vmunix.save exists
10158 test -f /vmunix.save && saved=1
```

10159       Command substitution and redirections without command names both occur in subshells, but  
10160       they are not necessarily the same ones. For example, in:

```
10161 exec 3> file
10162 var=$(echo foo >&3) 3>&1
```

10163       it is unspecified whether **foo** is echoed to the file or to standard output.

## 10164       **Command Search and Execution**

10165       This description requires that the shell can execute shell scripts directly, even if the underlying  
10166       system does not support the common "#!" interpreter convention. That is, if file **foo** contains  
10167       shell commands and is executable, the following executes **foo**:

```
10168 ./foo
```

10169       The command search shown here does not match all historical implementations. A more typical  
10170       sequence has been:

- 10171       • Any built-in (special or regular)
- 10172       • Functions
- 10173       • Path search for executable files

10174       But there are problems with this sequence. Since the programmer has no idea in advance which  
10175       utilities might have been built into the shell, a function cannot be used to override portably a  
10176       utility of the same name. (For example, a function named *cd* cannot be written for many  
10177       historical systems.) Furthermore, the *PATH* variable is partially ineffective in this case, and only  
10178       a pathname with a slash can be used to ensure a specific executable file is invoked.

10179       After the *execve()* failure described, the shell normally executes the file as a shell script. Some  
10180       implementations, however, attempt to detect whether the file is actually a script and not an  
10181       executable from some other architecture. The method used by the KornShell is allowed by the  
10182       text that indicates non-text files may be bypassed.

10183       The sequence selected for the Shell and Utilities volume of IEEE Std 1003.1-2001 acknowledges  
10184       that special built-ins cannot be overridden, but gives the programmer full control over which  
10185       versions of other utilities are executed. It provides a means of suppressing function lookup (via  
10186       the *command* utility) for the user's own functions and ensures that any regular built-ins or  
10187       functions provided by the implementation are under the control of the path search. The  
10188       mechanisms for associating built-ins or functions with executable files in the path are not  
10189       specified by the Shell and Utilities volume of IEEE Std 1003.1-2001, but the wording requires that  
10190       if either is implemented, the application is not able to distinguish a function or built-in from an  
10191       executable (other than in terms of performance, presumably). The implementation ensures that  
10192       all effects specified by the Shell and Utilities volume of IEEE Std 1003.1-2001 resulting from the  
10193       invocation of the regular built-in or function (interaction with the environment, variables, traps,  
10194       and so on) are identical to those resulting from the invocation of an executable file.

**Examples**

Consider three versions of the *ls* utility:

1. The application includes a shell function named *ls*.
2. The user writes a utility named *ls* and puts it in **/fred/bin**.
3. The example implementation provides *ls* as a regular shell built-in that is invoked (either by the shell or directly by *exec*) when the path search reaches the directory **/posix/bin**.

If *PATH*=**/posix/bin**, various invocations yield different versions of *ls*:

| Invocation                                             | Version of <i>ls</i> |
|--------------------------------------------------------|----------------------|
| <i>ls</i> (from within application script)             | (1) function         |
| <i>command ls</i> (from within application script)     | (3) built-in         |
| <i>ls</i> (from within makefile called by application) | (3) built-in         |
| <i>system("ls")</i>                                    | (3) built-in         |
| <i>PATH="/fred/bin:\$PATH" ls</i>                      | (2) user's version   |

**C.2.9.2 Pipelines**

Because pipeline assignment of standard input or standard output or both takes place before redirection, it can be modified by redirection. For example:

```
$ command1 2>&1 | command2
```

sends both the standard output and standard error of *command1* to the standard input of *command2*.

The reserved word **!** allows more flexible testing using AND and OR lists.

It was suggested that it would be better to return a non-zero value if any command in the pipeline terminates with non-zero status (perhaps the bitwise-inclusive OR of all return values). However, the choice of the last-specified command semantics are historical practice and would cause applications to break if changed. An example of historical behavior:

```
$ sleep 5 | (exit 4)
$ echo $?
4
$ (exit 4) | sleep 5
$ echo $?
0
```

**C.2.9.3 Lists**

The equal precedence of "&&" and "||" is historical practice. The standard developers evaluated the model used more frequently in high-level programming languages, such as C, to allow the shell logical operators to be used for complex expressions in an unambiguous way, but they could not allow historical scripts to break in the subtle way unequal precedence might cause. Some arguments were posed concerning the "{" or "(" groupings that are required historically. There are some disadvantages to these groupings:

- The "(" can be expensive, as they spawn other processes on some implementations. This performance concern is primarily an implementation issue.
- The "{" braces are not operators (they are reserved words) and require a trailing space after each '{', and a semicolon before each '}'. Most programmers (and certainly

10237 interactive users) have avoided braces as grouping constructs because of the problematic  
 10238 syntax required. Braces were not changed to operators because that would generate  
 10239 compatibility issues even greater than the precedence question; braces appear outside the  
 10240 context of a keyword in many shell scripts.

10241 IEEE PASC Interpretation 1003.2 #204 is applied, clarifying that the operators "&&" and "| |"  
 10242 are evaluated with left associativity.

### 10243 **Asynchronous Lists**

10244 The grammar treats a construct such as:

```
10245 foo & bar & bam &
```

10246 as one “asynchronous list”, but since the status of each element is tracked by the shell, the term  
 10247 “element of an asynchronous list” was introduced to identify just one of the **foo**, **bar**, or **bam**  
 10248 portions of the overall list.

10249 Unless the implementation has an internal limit, such as {CHILD\_MAX}, on the retained process  
 10250 IDs, it would require unbounded memory for the following example:

```
10251 while true
10252 do foo & echo $!
10253 done
```

10254 The treatment of the signals SIGINT and SIGQUIT with asynchronous lists is described in the  
 10255 Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.11, Signals and Error Handling.

10256 Since the connection of the input to the equivalent of **/dev/null** is considered to occur before  
 10257 redirections, the following script would produce no output:

```
10258 exec < /etc/passwd
10259 cat <&0 &
10260 wait
```

### 10261 **Sequential Lists**

10262 There is no additional rationale provided for this section.

### 10263 **AND Lists**

10264 There is no additional rationale provided for this section.

### 10265 **OR Lists**

10266 There is no additional rationale provided for this section.



## 10267 C.2.9.4 Compound Commands

10268 **Grouping Commands**

10269 The semicolon shown in `{compound-list;}` is an example of a control operator delimiting the `}`  
 10270 reserved word. Other delimiters are possible, as shown in the Shell and Utilities volume of  
 10271 IEEE Std 1003.1-2001, Section 2.10, Shell Grammar; `<newline>` is frequently used.

10272 A proposal was made to use the **<do-done>** construct in all cases where command grouping in  
 10273 the current process environment is performed, identifying it as a construct for the grouping  
 10274 commands, as well as for shell functions. This was not included because the shell already has a  
 10275 grouping construct for this purpose ("`{ }`"), and changing it would have been counter-  
 10276 productive.

10277 **For Loop**

10278 The format is shown with generous usage of `<newline>`s. See the grammar in the Shell and  
 10279 Utilities volume of IEEE Std 1003.1-2001, Section 2.10, Shell Grammar for a precise description of  
 10280 where `<newline>`s and semicolons can be interchanged.

10281 Some historical implementations support '`{`' and '`}`' as substitutes for **do** and **done**. The  
 10282 standard developers chose to omit them, even as an obsolescent feature. (Note that these  
 10283 substitutes were only for the **for** command; the **while** and **until** commands could not use them  
 10284 historically because they are followed by compound-lists that may contain "`{ ... }`" grouping  
 10285 commands themselves.)

10286 The reserved word pair **do ... done** was selected rather than **do ... od** (which would have  
 10287 matched the spirit of **if ... fi** and **case ... esac**) because *od* is already the name of a standard  
 10288 utility.

10289 PASC Interpretation 1003.2 #169 has been applied changing the grammar.

10290 **Case Conditional Construct**

10291 An optional left parenthesis before *pattern* was added to allow numerous historical KornShell  
 10292 scripts to conform. At one time, using the leading parenthesis was required if the **case** statement  
 10293 was to be embedded within a "`$ ( )`" command substitution; this is no longer the case with the  
 10294 POSIX shell. Nevertheless, many historical scripts use the left parenthesis, if only because it  
 10295 makes matching-parenthesis searching easier in *vi* and other editors. This is a relatively simple  
 10296 implementation change that is upwards-compatible for all scripts.

10297 Consideration was given to requiring *break* inside the *compound-list* to prevent falling through to  
 10298 the next pattern action list. This was rejected as being nonexistent practice. An interesting  
 10299 undocumented feature of the KornShell is that using "`;&`" instead of "`;;`" as a terminator  
 10300 causes the exact opposite behavior—the flow of control continues with the next *compound-list*.

10301 The pattern '`*`', given as the last pattern in a **case** construct, is equivalent to the default case in  
 10302 a C-language **switch** statement.

10303 The grammar shows that reserved words can be used as patterns, even if one is the first word on  
 10304 a line. Obviously, the reserved word **esac** cannot be used in this manner.

10305 **If Conditional Construct**

10306 The precise format for the command syntax is described in the Shell and Utilities volume of  
 10307 IEEE Std 1003.1-2001, Section 2.10, Shell Grammar.

10308 **While Loop**

10309 The precise format for the command syntax is described in the Shell and Utilities volume of  
 10310 IEEE Std 1003.1-2001, Section 2.10, Shell Grammar.

10311 **Until Loop**

10312 The precise format for the command syntax is described in the Shell and Utilities volume of  
 10313 IEEE Std 1003.1-2001, Section 2.10, Shell Grammar.

10314 C.2.9.5 *Function Definition Command*

10315 The description of functions in an early proposal was based on the notion that functions should  
 10316 behave like miniature shell scripts; that is, except for sharing variables, most elements of an  
 10317 execution environment should behave as if they were a new execution environment, and  
 10318 changes to these should be local to the function. For example, traps and options should be reset  
 10319 on entry to the function, and any changes to them do not affect the traps or options of the caller.  
 10320 There were numerous objections to this basic idea, and the opponents asserted that functions  
 10321 were intended to be a convenient mechanism for grouping common commands that were to be  
 10322 executed in the current execution environment, similar to the execution of the *dot* special  
 10323 built-in.

10324 It was also pointed out that the functions described in that early proposal did not provide a local  
 10325 scope for everything a new shell script would, such as the current working directory, or *umask*,  
 10326 but instead provided a local scope for only a few select properties. The basic argument was that  
 10327 if a local scope is needed for the execution environment, the mechanism already existed: the  
 10328 application can put the commands in a new shell script and call that script. All historical shells  
 10329 that implemented functions, other than the KornShell, have implemented functions that operate  
 10330 in the current execution environment. Because of this, traps and options have a global scope  
 10331 within a shell script. Local variables within a function were considered and included in another  
 10332 early proposal (controlled by the special built-in *local*), but were removed because they do not fit  
 10333 the simple model developed for functions and because there was some opposition to adding yet  
 10334 another new special built-in that was not part of historical practice. Implementations should  
 10335 reserve the identifier *local* (as well as *typeset*, as used in the KornShell) in case this local variable  
 10336 mechanism is adopted in a future version of IEEE Std 1003.1-2001.

10337 A separate issue from the execution environment of a function is the availability of that function  
 10338 to child shells. A few objectors maintained that just as a variable can be shared with child shells  
 10339 by exporting it, so should a function. In early proposals, the *export* command therefore had a *-f*  
 10340 flag for exporting functions. Functions that were exported were to be put into the environment  
 10341 as *name()*=*value* pairs, and upon invocation, the shell would scan the environment for these and  
 10342 automatically define these functions. This facility was strongly opposed and was omitted. Some  
 10343 of the arguments against exportable functions were as follows:

- 10344 • There was little historical practice. The Ninth Edition shell provided them, but there was  
 10345 controversy over how well it worked.
- 10346 • There are numerous security problems associated with functions appearing in the  
 10347 environment of a user and overriding standard utilities or the utilities owned by the  
 10348 application.

- There was controversy over requiring *make* to import functions, where it has historically used an *exec* function for many of its command line executions.

- Functions can be big and the environment is of a limited size. (The counter-argument was that functions are no different from variables in terms of size: there can be big ones, and there can be small ones—and just as one does not export huge variables, one does not export huge functions. However, this might not apply to the average shell-function writer, who typically writes much larger functions than variables.)

As far as can be determined, the functions in the Shell and Utilities volume of IEEE Std 1003.1-2001 match those in System V. Earlier versions of the KornShell had two methods of defining functions:

```
function fname { compound-list }
```

and:

```
fname() { compound-list }
```

The latter used the same definition as the Shell and Utilities volume of IEEE Std 1003.1-2001, but differed in semantics, as described previously. The current edition of the KornShell aligns the latter syntax with the Shell and Utilities volume of IEEE Std 1003.1-2001 and keeps the former as is.

The name space for functions is limited to that of a *name* because of historical practice. Complications in defining the syntactic rules for the function definition command and in dealing with known extensions such as the "@()" usage in the KornShell prevented the name space from being widened to a *word*. Using functions to support synonyms such as the "!!" and '% ' usage in the C shell is thus disallowed to conforming applications, but acceptable as an extension. For interactive users, the aliasing facilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 should be adequate for this purpose. It is recognized that the name space for utilities in the file system is wider than that currently supported for functions, if the portable filename character set guidelines are ignored, but it did not seem useful to mandate extensions in systems for so little benefit to conforming applications.

The "()" in the function definition command consists of two operators. Therefore, intermixing <blank>s with the *fname*, '(', and ')' is allowed, but unnecessary.

An example of how a function definition can be used wherever a simple command is allowed:

```
If variable i is equal to "yes",
define function foo to be ls -l
#
["$i" = yes] && foo() {
 ls -l
}
```

## C.2.10 Shell Grammar

There are several subtle aspects of this grammar where conventional usage implies rules about the grammar that in fact are not true.

For *compound\_list*, only the forms that end in a *separator* allow a reserved word to be recognized, so usually only a *separator* can be used where a compound list precedes a reserved word (such as **Then**, **Else**, **Do**, and **Rbrace**). Explicitly requiring a separator would disallow such valid (if rare) statements as:

```
if (false) then (echo x) else (echo y) fi
```

- 10393 See the Note under special grammar rule (1).
- 10394 Concerning the third sentence of rule (1) (“Also, if the parser ...”):
- 10395 • This sentence applies rather narrowly: when a compound list is terminated by some clear  
10396 delimiter (such as the closing **fi** of an inner **if\_clause**) then it would apply; where the  
10397 compound list might continue (as in after a ‘;’), rule (7a) (and consequently the first  
10398 sentence of rule (1)) would apply. In many instances the two conditions are identical, but this  
10399 part of rule (1) does not give license to treating a **WORD** as a reserved word unless it is in a  
10400 place where a reserved word has to appear.
  - 10401 • The statement is equivalent to requiring that when the LR(1) lookahead set contains exactly  
10402 one reserved word, it must be recognized if it is present. (Here “LR(1)” refers to the  
10403 theoretical concepts, not to any real parser generator.)
- 10404 For example, in the construct below, and when the parser is at the point marked with ‘^’,  
10405 the only next legal token is **then** (this follows directly from the grammar rules):
- ```
10406     if if...fi then ... fi
10407         ^
```
- 10408 At that point, the **then** must be recognized as a reserved word.
- 10409 (Depending on the parser generator actually used, “extra” reserved words may be in some
10410 lookahead sets. It does not really matter if they are recognized, or even if any possible
10411 reserved word is recognized in that state, because if it is recognized and is not in the
10412 (theoretical) LR(1) lookahead set, an error is ultimately detected. In the example above, if
10413 some other reserved word (for example, **while**) is also recognized, an error occurs later.
- 10414 This is approximately equivalent to saying that reserved words are recognized after other
10415 reserved words (because it is after a reserved word that this condition occurs), but avoids the
10416 “except for ...” list that would be required for **case**, **for**, and so on. (Reserved words are of
10417 course recognized anywhere a *simple_command* can appear, as well. Other rules take care of
10418 the special cases of non-recognition, such as rule (4) for **case** statements.)
- 10419 Note that the body of here-documents are handled by token recognition (see the Shell and
10420 Utilities volume of IEEE Std 1003.1-2001, Section 2.3, Token Recognition) and do not appear in
10421 the grammar directly. (However, the here-document I/O redirection operator is handled as part
10422 of the grammar.)
- 10423 The start symbol of the grammar (**complete_command**) represents either input from the
10424 command line or a shell script. It is repeatedly applied by the interpreter to its input and
10425 represents a single “chunk” of that input as seen by the interpreter.

10426 C.2.10.1 Shell Grammar Lexical Conventions

10427 There is no additional rationale provided for this section.

10428 C.2.10.2 Shell Grammar Rules

10429 There is no additional rationale provided for this section.

10430 **C.2.11 Signals and Error Handling**

10431 There is no additional rationale provided for this section.

10432 **C.2.12 Shell Execution Environment**

10433 Some implementations have implemented the last stage of a pipeline in the current environment
10434 so that commands such as:

10435 `command | read foo`

10436 set variable **foo** in the current environment. This extension is allowed, but not required;
10437 therefore, a shell programmer should consider a pipeline to be in a subshell environment, but
10438 not depend on it.

10439 In early proposals, the description of execution environment failed to mention that each
10440 command in a multiple command pipeline could be in a subshell execution environment. For
10441 compatibility with some historical shells, the wording was phrased to allow an implementation
10442 to place any or all commands of a pipeline in the current environment. However, this means that
10443 a POSIX application must assume each command is in a subshell environment, but not depend
10444 on it.

10445 The wording about shell scripts is meant to convey the fact that describing “trap actions” can
10446 only be understood in the context of the shell command language. Outside of this context, such
10447 as in a C-language program, signals are the operative condition, not traps.

10448 **C.2.13 Pattern Matching Notation**

10449 Pattern matching is a simpler concept and has a simpler syntax than REs, as the former is
10450 generally used for the manipulation of filenames, which are relatively simple collections of
10451 characters, while the latter is generally used to manipulate arbitrary text strings of potentially
10452 greater complexity. However, some of the basic concepts are the same, so this section points
10453 liberally to the detailed descriptions in the Base Definitions volume of IEEE Std 1003.1-2001,
10454 Chapter 9, Regular Expressions.

10455 *C.2.13.1 Patterns Matching a Single Character*

10456 Both quoting and escaping are described here because pattern matching must work in three
10457 separate circumstances:

1. Calling directly upon the shell, such as in pathname expansion or in a **case** statement. All
10459 of the following match the string or file **abc**:

10460 `abc "abc" a"b" c a\bc a[b] c a["b"] c a[\b] c a["\b"] c a?c a*c`

10461 The following do not:

10462 `"a?c" a*c a\[b] c`

2. Calling a utility or function without going through a shell, as described for *find* and the
10464 *fnmatch()* function defined in the System Interfaces volume of IEEE Std 1003.1-2001.

3. Calling utilities such as *find*, *cpio*, *tar*, or *pax* through the shell command line. In this case,
10466 shell quote removal is performed before the utility sees the argument. For example, in:

10467 `find /bin -name "e\c[\h]o" -print`

10468 after quote removal, the backslashes are presented to *find* and it treats them as escape
10469 characters. Both precede ordinary characters, so the *c* and *h* represent themselves and *echo*
10470 would be found on many historical systems (that have it in **/bin**). To find a filename that

10471 contained shell special characters or pattern characters, both quoting and escaping are
10472 required, such as:

10473 `pax -r . . . "*a\(\?"`

10474 to extract a filename ending with "a(?".

10475 Conforming applications are required to quote or escape the shell special characters (sometimes
10476 called metacharacters). If used without this protection, syntax errors can result or
10477 implementation extensions can be triggered. For example, the KornShell supports a series of
10478 extensions based on parentheses in patterns.

10479 The restriction on a circumflex in a bracket expression is to allow implementations that support
10480 pattern matching using the circumflex as the negation character in addition to the exclamation
10481 mark. A conforming application must use something like "[\^!]" to match either character.

10482 C.2.13.2 Patterns Matching Multiple Characters

10483 Since each asterisk matches zero or more occurrences, the patterns "a*b" and "a**b" have
10484 identical functionality.

10485 Examples

10486 `a[bc]` Matches the strings "ab" and "ac".

10487 `a*d` Matches the strings "ad", "abd", and "abcd", but not the string "abc".

10488 `a*d*` Matches the strings "ad", "abcd", "abcdef", "aaaad", and "adddd".

10489 `*a*d` Matches the strings "ad", "abcd", "efabcd", "aaaad", and "adddd".

10490 C.2.13.3 Patterns Used for Filename Expansion

10491 The caveat about a slash within a bracket expression is derived from historical practice. The
10492 pattern "a[b/c]d" does not match such pathnames as **abd** or **a/d**. On some implementations
10493 (including those conforming to the Single UNIX Specification), it matched a pathname of
10494 literally "a[b/c]d". On other systems, it produced an undefined condition (an unescaped '['
10495 used outside a bracket expression). In this version, the XSI behavior is now required.

10496 Filenames beginning with a period historically have been specially protected from view on
10497 UNIX systems. A proposal to allow an explicit period in a bracket expression to match a leading
10498 period was considered; it is allowed as an implementation extension, but a conforming
10499 application cannot make use of it. If this extension becomes popular in the future, it will be
10500 considered for a future version of the Shell and Utilities volume of IEEE Std 1003.1-2001.

10501 Historical systems have varied in their permissions requirements. To match **f*/bar** has required
10502 read permissions on the **f*** directories in the System V shell, but the Shell and Utilities volume of
10503 IEEE Std 1003.1-2001, the C shell, and KornShell require only search permissions.

10504 **C.2.14 Special Built-In Utilities**

10505 See the RATIONALE sections on the individual reference pages.

10506 **C.3 Batch Environment Services and Utilities**10507 **Scope of the Batch Environment Services and Utilities Option**

10508 This section summarizes the deliberations of the IEEE P1003.15 (Batch Environment) working
 10509 group in the development of the Batch Environment Services and Utilities option, which covers
 10510 a set of services and utilities defining a batch processing system.

10511 This informative section contains historical information concerning the contents of the
 10512 amendment and describes why features were included or discarded by the working group.

10513 **History of Batch Systems**

10514 The supercomputing technical committee began as a “Birds Of a Feather” (BOF) at the January
 10515 1987 Usenix meeting. There was enough general interest to form a supercomputing attachment
 10516 to the /usr/group working groups. Several subgroups rapidly formed. Of those subgroups, the
 10517 batch group was the most ambitious. The first early meetings were spent evaluating user needs
 10518 and existing batch implementations.

10519 To evaluate user needs, individuals from the supercomputing community came and presented
 10520 their needs. Common requests were flexibility, interoperability, control of resources, and ease-
 10521 of-use. Backward-compatibility was not an issue. The working group then evaluated some
 10522 existing systems. The following different systems were evaluated:

- 10523 • PROD
- 10524 • Convex Distributed Batch
- 10525 • NQS
- 10526 • CTSS
- 10527 • MDQS from Ballistics Research Laboratory (BRL)

10528 Finally, NQS was chosen as a model because it satisfied not only the most user requirements, but
 10529 because it was public domain, already implemented on a variety of hardware platforms, and
 10530 network-based.

10531 **Historical Implementations of Batch Systems**

10532 Deferred processing of work under the control of a scheduler has been a feature of most
 10533 proprietary operating systems from the earliest days of multi-user systems in order to maximize
 10534 utilization of the computer.

10535 The arrival of UNIX systems proved to be a dilemma to many hardware providers and users
 10536 because it did not include the sophisticated batch facilities offered by the proprietary systems.
 10537 This omission was rectified in 1986 by NASA Ames Research Center who developed the
 10538 Network Queuing System (NQS) as a portable UNIX application that allowed the routing and
 10539 processing of batch “jobs” in a network. To encourage its usage, the product was later put into
 10540 the public domain. It was promptly picked up by UNIX hardware providers, and ported and
 10541 developed for their respective hardware and UNIX implementations.

Many major vendors, who traditionally offer a batch-dominated environment, ported the public-domain product to their systems, customized it to support the capabilities of their systems, and added many customer-requested features.

Due to the strong hardware provider and customer acceptance of NQS, it was decided to use NQS as the basis for the POSIX Batch Environment amendment in 1987. Other batch systems considered at the time included CTSS, MDQS (a forerunner of NQS from the Ballistics Research Laboratory), and PROD (a Los Alamos Labs development). None were thought to have both the functionality and acceptability of NQS.

10550 **NQS Differences from the *at* utility**

The base standard *at* and *batch* utilities are not sufficient to meet the batch processing needs in a supercomputing environment and additional functionality in the areas of resource management, job scheduling, system management, and control of output is required.

10554 **Batch Environment Services and Utilities Option Definitions**

The concept of a batch job is closely related to a session with a session leader. The main difference is that a batch job does not have a controlling terminal. There has been much debate over whether to use the term “request” or “job”. Job was the final choice because of the historical use of this term in the batch environment.

The current definition for job identifiers is not sufficient with the model of destinations. The current definition is:

```
sequence_number.originating_host
```

Using the model of destination, a host may include multiple batch nodes, the location of which is identified uniquely by a name or directory service. If the current definition is used, batch nodes running on the same host would have to coordinate their use of sequence numbers, as sequence numbers are assigned by the originating host. The alternative is to use the originating batch node name instead of the originating host name.

The reasons for wishing to run more than one batch system per host could be the following.

A test and production batch system are maintained on a single host. This is most likely in a development facility, but could also arise when a site is moving from one version to another. The new batch system could be installed as a test version that is completely separate from the production batch system, so that problems can be isolated to the test system. Requiring the batch nodes to coordinate their use of sequence numbers creates a dependency between the two nodes, and that defeats the purpose of running two nodes.

A site has multiple departments using a single host, with different management policies. An example of contention might be in job selection algorithms. One group might want a FIFO type of selection, while another group wishes to use a more complex algorithm based on resource availability. Again, requiring the batch nodes to coordinate is an unnecessary binding.

The proposal eventually accepted was to replace originating host with originating batch node. This supplies sufficient granularity to ensure unique job identifiers. If more than one batch node is on a particular host, they each have their own unique name.

The queue portion of a destination is not part of the job identifier as these are not required to be unique between batch nodes. For instance, two batch nodes may both have queues called small, medium, and large. It is only the batch node name that is uniquely identifiable throughout the batch system. The queue name has no additional function in this context.

10585 Assume there are three batch nodes, each of which has its own name server. On batch node one,
10586 there are no queues. On batch node two, there are fifty queues. On batch node three, there are
10587 forty queues. The system administrator for batch node one does not have to configure queues,
10588 because there are none implemented. However, if a user wishes to send a job to either batch
10589 node two or three, the system administrator for batch node one must configure a destination
10590 that maps to the appropriate batch node and queue. If every queue is to be made accessible from
10591 batch node one, the system administrator has to configure ninety destinations.

10592 To avoid requiring this, there should be a mechanism to allow a user to separate the destination
10593 into a batch node name and a queue name. Then, an implementation that is configured to get to
10594 all the batch nodes does not need any more configuration to allow a user to get to all of the
10595 queues on all of the batch nodes. The node name is used to locate the batch node, while the
10596 queue name is sent unchanged to that batch node.

10597 The following are requirements that a destination identifier must be capable of providing:

- 10598 • The ability to direct a job to a queue in a particular batch node.
- 10599 • The ability to direct a job to a particular batch node.
- 10600 • The ability to group at a higher level than just one queue. This includes grouping similar
10601 queues across multiple batch nodes (this is a pipe queue).
- 10602 • The ability to group batch nodes. This allows a user to submit a job to a group name with no
10603 knowledge of the batch node configuration. This also provides aliasing as a special case.
10604 Aliasing is a group containing only one batch node name. The group name is the alias.

10605 In addition, the administrator has the following requirements:

- 10606 • The ability to control access to the queues.
- 10607 • The ability to control access to the batch nodes.
- 10608 • The ability to control access to groups of queues (pipe queues).
- 10609 • The ability to configure retry time intervals and durations.

10610 The requirements of the user are met by destination as explained in the following.

10611 The user has the ability to specify a queue name, which is known only to the batch node
10612 specified. There is no configuration of these queues required on the submitting node.

10613 The user has the ability to specify a batch node whose name is network-unique. The
10614 configuration required is that the batch node be defined as an application, just as other
10615 applications such as FTP are configured.

10616 Once a job reaches a queue, it can again become a user of the batch system. The batch node can
10617 choose to send the job to another batch node or queue or both. In other words, the routing is at
10618 an application level, and it is up to the batch system to choose where the job will be sent.
10619 Configuration is up to the batch node where the queue resides. This provides grouping of
10620 queues across batch nodes or within a batch node. The user submits the job to a queue, which by
10621 definition routes the job to other queues or nodes or both.

10622 A node name may be given to a naming service, which returns multiple addresses as opposed to
10623 just one. This provides grouping at a batch node level. This is a local issue, meaning that the
10624 batch node must choose only one of these addresses. The list of addresses is not sent with the
10625 job, and once the job is accepted on another node, there is no connection between the list and the
10626 job. The requirements of the administrator are met by destination as explained in the following.

10627 The control of queues is a batch system issue, and will be done using the batch administrative
10628 utilities.

10629 The control of nodes is a network issue, and will be done through whatever network facilities
10630 are available.

10631 The control of access to groups of queues (pipe queues) is covered by the control of any other
10632 queue. The fact that the job may then be sent to another destination is not relevant.

10633 The propagation of a job across more than one point-to-point connection was dropped because
10634 of its complexity and because all of the issues arising from this capability could not be resolved.
10635 It could be provided as additional functionality at some time in the future.

10636 The addition of *network* as a defined term was done to clarify the difference between a network
10637 of batch nodes as opposed to a network of hosts. A network of batch nodes is referred to as a
10638 batch system. The network refers to the actual host configuration. A single host may have
10639 multiple batch nodes.

10640 In the absence of a standard network naming convention, this option establishes its own
10641 convention for the sake of consistency and expediency. This is subject to change, should a future
10642 working group develop a standard naming convention for network pathnames.

10643 C.3.1 Batch General Concepts

10644 During the development of the Batch Environment Services and Utilities option, a number of
10645 topics were discussed at length which influenced the wording of the normative text but could
10646 not be included in the final text. The following items are some of the most significant terms and
10647 concepts of those discussed:

- 10648 • Small and Consistent Command Set

10649 Often, conventional utilities from UNIX systems have a very complicated utility syntax and
10650 usage. This can often result in confusion and errors when trying to use them. The Batch
10651 Environment Services and Utilities option utility set, on the other hand, has been paired to a
10652 small set of robust utilities with an orthogonal calling sequence.

- 10653 • Checkpoint/Restart

10654 This feature permits an already executing process to checkpoint or save its contents. Some
10655 implementations permit this at both the batch utility level (for example, checkpointing this
10656 job upon its abnormal termination) or from within the job itself via a system call. Support of
10657 checkpoint/restart is optional. A conscious, careful effort was made to make the *qsub* utility
10658 consistently refer to checkpoint/restart as optional functionality.

- 10659 • Rerunability

10660 When a user submits a job for batch processing, they can designate it “rerunnable” in that it
10661 will automatically resume execution from the start of the job if the machine on which it was
10662 executing crashes for some reason. The decision on whether the job will be rerun or not is
10663 entirely up to the submitter of the job and no decisions will be made within the batch system.
10664 A job that is rerunnable and has been submitted with the proper checkpoint/restart switch
10665 will first be checkpointed and execution begun from that point. Furthermore, use of the
10666 implementation-defined checkpoint/restart feature will not be defined in this context.

- 10667 • Error Codes

10668 All utilities exit with error status zero (0) if successful, one (1) if a user error occurred, and
10669 two (2) for an internal Batch Environment Services and Utilities option error.

- 10670 • Level of Portability

10671 Portability is specified at both the user, operator, and administrator levels. A conforming
10672 batch implementation prevents identical functionality and behavior at all these levels.

10673	Additionally, portable batch shell scripts with embedded Batch Environment Services and	
10674	Utilities option utilities add an additional level of portability.	
10675	• Resource Specification	
10676	A small set of globally understood resources, such as memory and CPU time, is specified. All	
10677	conforming batch implementations are able to process them in a manner consistent with the	
10678	yet-to-be-developed resource management model. Resources not in this amendment set are	
10679	ignored and passed along as part of the argument stream of the utility.	
10680	• Queue Position	
10681	Queue position is the place a job occupies in a queue. It is dependent on a variety of factors	
10682	such as submission time and priority. Since priority may be affected by the implementation	
10683	of fair share scheduling, the definition of queue position is implementation-defined.	
10684	• Queue ID	
10685	A numerical queue ID is an external requirement for purposes of accounting. The	
10686	identification number was chosen over queue name for processing convenience.	
10687	• Job ID	
10688	A common notion of “jobs” is a collection of processes whose process group cannot be	
10689	altered and is used for resource management and accounting. This concept is	
10690	implementation-defined and, as such, has been omitted from the batch amendment.	
10691	• Bytes <i>versus</i> Words	
10692	Except for one case, bytes are used as the standard unit for memory size. Furthermore, the	
10693	definition of a word varies from machine to machine. Therefore, bytes will be the default unit	
10694	of memory size.	
10695	• Regular Expressions	
10696	The standard definition of regular expressions is much too broad to be used in the batch	
10697	utility syntax. All that is needed is a simple concept of “all”; for example, delete all my jobs	
10698	from the named queue. For this reason, regular expressions have been eliminated from the	
10699	batch amendment.	
10700	• Display Privacy	
10701	How much data should be displayed locally through functions? Local policy dictates the	
10702	amount of privacy. Library functions must be used to create and enforce local policy.	
10703	Network and local <i>qstats</i> must reflect the policy of the server machine.	
10704	• Remote Host Naming Convention	
10705	It was decided that host names would be a maximum of 255 characters in length, with at	
10706	most 15 characters being shown in displays. The 255 character limit was chosen because it is	
10707	consistent with BSD. The 15-character limit was an arbitrary decision.	
10708	• Network Administration	
10709	Network administration is important, but is outside the scope of the batch amendment.	
10710	Network administration could be done with <i>rsh</i> . However, authentication becomes two-	
10711	sided.	
10712	• Network Administration Philosophy	
10713	Keep it simple. Centralized management should be possible. For example, Los Alamos needs	
10714	a dumb set of CPUs to be managed by a central system <i>versus</i> several independently-	

10715 managed systems as is the general case for the Batch Environment Services and Utilities
10716 option.

- 10717 • Operator Utility Defaults (that is, Default Host, User, Account, and so on)

10718 It was decided that usability would override orthogonality and syntactic consistency.

- 10719 • The Batch System Manager and Operator Distinction

10720 The distinction between manager and operator is that operators can only control the flow of
10721 jobs. A manager can alter the batch system configuration in addition to job flow. POSIX
10722 makes a distinction between user and system administrator but goes no further. The
10723 concepts of manager and operator privileges fall under local policy. The distinction between
10724 manager and operator is historical in batch environments, and the Batch Environment
10725 Services and Utilities option has continued that distinction.

- 10726 • The Batch System Administrator

10727 An administrator is equivalent to a batch system manager.

10728 C.3.2 Batch Services

10729 This rationale is provided as informative rather than normative text, to avoid placing
10730 requirements on implementors regarding the use of symbolic constants, but at the same time to
10731 give implementors a preferred practice for assigning values to these constants to promote
10732 interoperability.

10733 The *Checkpoint* and *Minimum_Cpu_Interval* attributes induce a variety of behavior depending
10734 upon their values. Some jobs cannot or should not be checkpointed. Other users will simply
10735 need to ensure job continuation across planned downtimes; for example, scheduled preventive
10736 maintenance. For users consuming expensive resources, or for jobs that run longer than the
10737 mean time between failures, however, periodic checkpointing may be essential. However,
10738 system administrators must be able to set minimum checkpoint intervals on a queue-by-queue
10739 basis to guard against, for example, naive users specifying interval values too small on
10740 memory-intensive jobs. Otherwise, system overhead would adversely affect performance.

10741 The use of symbolic constants, such as `NO_CHECKPOINT`, was introduced to lend a degree of
10742 formalism and portability to this option.

10743 Support for checkpointing is optional for servers. However, clients must provide for the `-c`
10744 option, since in a distributed environment the job may run on a server that does provide such
10745 support, even if the host of the client does not support the checkpoint feature.

10746 If the user does not specify the `-c` option, the default action is left unspecified by this option.
10747 Some implementations may wish to do checkpointing by default; others may wish to checkpoint
10748 only under an explicit request from the user.

10749 The *Priority* attribute has been made non-optional. All clients already had been required to
10750 support the `-p` option. The concept of prioritization is common in historical implementations.
10751 The default priority is left to the server to establish.

10752 The *Hold_Types* attribute has been modified to allow for implementation-defined hold types to
10753 be passed to a batch server.

10754 It was the intent of the IEEE P1003.15 working group to mandate the support for the
10755 *Resource_List* attribute in this option by referring to another amendment, specifically the
10756 IEEE P1003.1a draft standard. However, during the development of the IEEE P1003.1a draft
10757 standard this was excluded. As such this requirement has been removed from the normative
10758 text.

10759 The *Shell_Path* attribute has been modified to accept a list of shell paths that are associated with
 10760 a host. The name of the attribute has been changed to *Shell_Path_List*.

10761 **C.3.3 Common Behavior for Batch Environment Utilities**

10762 This section was defined to meet the goal of a “Small and Consistent Command Set” for this
 10763 option.

10764 **C.4 Utilities**

10765 For the utilities included in IEEE Std 1003.1-2001, see the RATIONALE sections on the individual
 10766 reference pages.

10767 **Exclusion of Utilities**

10768 The set of utilities contained in IEEE Std 1003.1-2001 is drawn from the base documents, with
 10769 one addition: the *c99* utility. This section contains rationale for some of the deliberations that led
 10770 to this set of utilities, and why certain utilities were excluded.

10771 Many utilities were evaluated by the standard developers; more historical utilities were
 10772 excluded from the base documents than included. The following list contains many common
 10773 UNIX system utilities that were not included as mandatory utilities, in the User Portability
 10774 Utilities option, in the XSI extension, or in one of the software development groups. It is
 10775 logistically difficult for this rationale to distribute correctly the reasons for not including a utility
 10776 among the various utility options. Therefore, this section covers the reasons for all utilities not
 10777 included in IEEE Std 1003.1-2001.

10778 This rationale is limited to a discussion of only those utilities actively or indirectly evaluated by
 10779 the standard developers of the base documents, rather than the list of all known UNIX utilities
 10780 from all its variants.

10781 *adb* The intent of the various software development utilities was to assist in the
 10782 installation (rather than the actual development and debugging) of applications.
 10783 This utility is primarily a debugging tool. Furthermore, many useful aspects of *adb*
 10784 are very hardware-specific.

10785 *as* Assemblers are hardware-specific and are included implicitly as part of the
 10786 compilers in IEEE Std 1003.1-2001.

10787 *banner* The only known use of this command is as part of the *lp* printer header pages. It
 10788 was decided that the format of the header is implementation-defined, so this utility
 10789 is superfluous to application portability.

10790 *calendar* This reminder service program is not useful to conforming applications.

10791 *cancel* The *lp* (line printer spooling) system specified is the most basic possible and did
 10792 not need this level of application control.

10793 *chroot* This is primarily of administrative use, requiring superuser privileges.

10794 *col* No utilities defined in IEEE Std 1003.1-2001 produce output requiring such a filter.
 10795 The *nroff* text formatter is present on many historical systems and will continue to
 10796 remain as an extension; *col* is expected to be shipped by all the systems that ship
 10797 *nroff*.

10798 *cpio* This has been replaced by *pax*, for reasons explained in the rationale for that utility.

10799	<i>cpp</i>	This is subsumed by <i>c99</i> .
10800	<i>cu</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
10801		
10802	<i>dc</i>	
10803		The functionality of this utility can be provided by the <i>bc</i> utility; <i>bc</i> was selected because it was easier to use and had superior functionality. Although the historical versions of <i>bc</i> are implemented using <i>dc</i> as a base, IEEE Std 1003.1-2001 prescribes the interface and not the underlying mechanism used to implement it.
10804		
10805		
10806	<i>dircmp</i>	
10807		Although a useful concept, the historical output of this directory comparison program is not suitable for processing in application programs. Also, the <i>diff -r</i> command gives equivalent functionality.
10808		
10809	<i>dis</i>	Disassemblers are hardware-specific.
10810	<i>emacs</i>	The community of <i>emacs</i> editing enthusiasts was adamant that the full <i>emacs</i> editor not be included in the base documents because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community. The author of the original <i>emacs</i> program also expressed his desire to omit the program. Furthermore, there were a number of historical UNIX systems that did not include <i>emacs</i> , or included it without supporting it, but there were very few that did not include and support <i>vi</i> .
10811		
10812		
10813		
10814		
10815		
10816		
10817		
10818	<i>ld</i>	This is subsumed by <i>c99</i> .
10819	<i>line</i>	The functionality of <i>line</i> can be provided with <i>read</i> .
10820	<i>lint</i>	This technology is partially subsumed by <i>c99</i> . It is also hard to specify the degree of checking for possible error conditions in programs in any compiler, and specifying what <i>lint</i> would do in these cases is equally difficult.
10821		
10822		It is fairly easy to specify what a compiler does. It requires specifying the language, what it does with that language, and stating that the interpretation of any incorrect program is unspecified. Unfortunately, any description of <i>lint</i> is required to specify what to do with erroneous programs. Since the number of possible errors and questionable programming practices is infinite, one cannot require <i>lint</i> to detect all errors of any given class.
10823		
10824		
10825		
10826		
10827		
10828		
10829		Additionally, some vendors complained that since many compilers are distributed in a binary form without a <i>lint</i> facility (because the ISO C standard does not require one), implementing the standard as a stand-alone product will be much harder. Rather than being able to build upon a standard compiler component (simply by providing <i>c99</i> as an interface), source to that compiler would most likely need to be modified to provide the <i>lint</i> functionality. This was considered a major burden on system providers for a very small gain to developers (users).
10830		
10831		
10832		
10833		
10834		
10835		
10836	<i>login</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
10837		
10838	<i>lorder</i>	This utility is an aid in creating an implementation-defined detail of object libraries that the standard developers did not feel required standardization.
10839		
10840	<i>lpstat</i>	The <i>lp</i> system specified is the most basic possible and did not need this level of application control.
10841		
10842	<i>mail</i>	This utility was omitted in favor of <i>mailx</i> because there was a considerable functionality overlap between the two.
10843		

10844	<i>mknod</i>	This was omitted in favor of <i>mkfifo</i> , as <i>mknod</i> has too many implementation-defined functions.
10845		
10846	<i>news</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
10847		
10848	<i>pack</i>	This compression program was considered inferior to <i>compress</i> .
10849	<i>passwd</i>	This utility was proposed in a historical draft of the base documents but met with too many objections to be included. There were various reasons: • Changing a password should not be viewed as a command, but as part of the login sequence. Changing a password should only be done while a trusted path is in effect. • Even though the text in early drafts was intended to allow a variety of implementations to conform, the security policy for one site may differ from another site running with identical hardware and software. One site might use password authentication while the other did not. Vendors could not supply a <i>passwd</i> utility that would conform to IEEE Std 1003.1-2001 for all sites using their system. • This is really a subject for a system administration working group or a security working group.
10850		
10851		
10852		
10853		
10854		
10855		
10856		
10857		
10858		
10859		
10860		
10861		
10862	<i>pcat</i>	This compression program was considered inferior to <i>zcat</i> .
10863	<i>pg</i>	This duplicated many of the features of the <i>more</i> pager, which was preferred by the standard developers.
10864		
10865	<i>prof</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
10866		
10867		
10868	RCS	RCS was originally considered as part of a version control utilities portion of the scope. However, this aspect was abandoned by the standard developers. SCCS is now included as an optional part of the XSI extension.
10869		
10870		
10871	<i>red</i>	Restricted editor. This was not considered by the standard developers because it never provided the level of security restriction required.
10872		
10873	<i>rsh</i>	Restricted shell. This was not considered by the standard developers because it does not provide the level of security restriction that is implied by historical documentation.
10874		
10875		
10876	<i>sdb</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool. Furthermore, some useful aspects of <i>sdb</i> are very hardware-specific.
10877		
10878		
10879		
10880	<i>sdiff</i>	The “side-by-side <i>diff</i> ” utility from System V was omitted because it is used infrequently, and even less so by conforming applications. Despite being in System V, it is not in the SVID or XPG.
10881		
10882		
10883	<i>shar</i>	Any of the numerous “shell archivers” were excluded because they did not meet the requirement of existing practice.
10884		
10885	<i>shl</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs. The job control aspects of the shell command language are generally more useful.
10886		
10887		

10888	<i>size</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
10889		
10890		
10891	<i>spell</i>	This utility is not useful from shell scripts or typical application programs. The <i>spell</i> utility was considered, but was omitted because there is no known technology that can be used to make it recognize general language for user-specified input without providing a complete dictionary along with the input file.
10892		
10893		
10894		
10895	<i>su</i>	This utility is not useful from shell scripts or typical application programs. (There was also sentiment to avoid security-related utilities.)
10896		
10897	<i>sum</i>	This utility was renamed <i>cksum</i> .
10898	<i>tar</i>	This has been replaced by <i>pax</i> , for reasons explained in the rationale for that utility.
10899	<i>unpack</i>	This compression program was considered inferior to <i>uncompress</i> .
10900	<i>wall</i>	This utility is terminal-oriented and is not useful in shell scripts or typical applications. It is generally used only by system administrators.
10901		

Rationale (Informative)

10902

10903

Part D:

10904

Portability Considerations

10905

The Open Group

10906

The Institute of Electrical and Electronics Engineers, Inc.

Portability Considerations (Informative)

This section contains information to satisfy various international requirements:

- Section D.1 describes perceived user requirements.
- Section D.2 (on page 270) indicates how the facilities of IEEE Std 1003.1-2001 satisfy those requirements.
- Section D.3 (on page 277) offers guidance to writers of profiles on how the configurable options, limits, and optional behavior of IEEE Std 1003.1-2001 should be cited in profiles.

D.1 User Requirements

This section describes the user requirements that were perceived by the developers of IEEE Std 1003.1-2001. The primary source for these requirements was an analysis of historical practice in widespread use, as typified by the base documents listed in Section A.1.1 (on page 3).

IEEE Std 1003.1-2001 addresses the needs of users requiring open systems solutions for source code portability of applications. It currently addresses users requiring open systems solutions for source-code portability of applications involving multi-programming and process management (creating processes, signaling, and so on); access to files and directories in a hierarchy of file systems (opening, reading, writing, deleting files, and so on); access to asynchronous communications ports and other special devices; access to information about other users of the system; facilities supporting applications requiring bounded (realtime) response.

The following users are identified for IEEE Std 1003.1-2001:

- Those employing applications written in high-level languages, such as C, Ada, or FORTRAN.
- Users who desire conforming applications that do not necessarily require the characteristics of high-level languages (for example, the speed of execution of compiled languages or the relative security of source code intellectual property inherent in the compilation process).
- Users who desire conforming applications that can be developed quickly and can be modified readily without the use of compilers and other system components that may be unavailable on small systems or those without special application development capabilities.
- Users who interact with a system to achieve general-purpose time-sharing capabilities common to most business or government offices or academic environments: editing, filing, inter-user communications, printing, and so on.
- Users who develop applications for POSIX-conformant systems.
- Users who develop applications for UNIX systems.

An acknowledged restriction on applicable users is that they are limited to the group of individuals who are familiar with the style of interaction characteristic of historically-derived systems based on one of the UNIX operating systems (as opposed to other historical systems with different models, such as MS/DOS, Macintosh, VMS, MVS, and so on). Typical users would include program developers, engineers, or general-purpose time-sharing users.

The requirements of users of IEEE Std 1003.1-2001 can be summarized as a single goal: *application source portability*. The requirements of the user are stated in terms of the requirements

10947 of portability of applications. This in turn becomes a requirement for a standardized set of
10948 syntax and semantics for operations commonly found on many operating systems.

10949 The following sections list the perceived requirements for application portability.

10950 **D.1.1 Configuration Interrogation**

10951 An application must be able to determine whether and how certain optional features are
10952 provided and to identify the system upon which it is running, so that it may appropriately adapt
10953 to its environment.

10954 Applications must have sufficient information to adapt to varying behaviors of the system.

10955 **D.1.2 Process Management**

10956 An application must be able to manage itself, either as a single process or as multiple processes.
10957 Applications must be able to manage other processes when appropriate.

10958 Applications must be able to identify, control, create, and delete processes, and there must be
10959 communication of information between processes and to and from the system.

10960 Applications must be able to use multiple flows of control with a process (threads) and
10961 synchronize operations between these flows of control.

10962 **D.1.3 Access to Data**

10963 Applications must be able to operate on the data stored on the system, access it, and transmit it
10964 to other applications. Information must have protection from unauthorized or accidental access
10965 or modification.

10966 **D.1.4 Access to the Environment**

10967 Applications must be able to access the external environment to communicate their input and
10968 results.

10969 **D.1.5 Access to Determinism and Performance Enhancements**

10970 Applications must have sufficient control of resource allocation to ensure the timeliness of
10971 interactions with external objects.

10972 **D.1.6 Operating System-Dependent Profile**

10973 The capabilities of the operating system may make certain optional characteristics of the base
10974 language in effect no longer optional, and this should be specified.

10975 **D.1.7 I/O Interaction**

10976 The interaction between the C language I/O subsystem (*stdio*) and the I/O subsystem of
10977 IEEE Std 1003.1-2001 must be specified.

10978 D.1.8 Internationalization Interaction

10979 The effects of the environment of IEEE Std 1003.1-2001 on the internationalization facilities of the
10980 C language must be specified.

10981 D.1.9 C-Language Extensions

10982 Certain functions in the C language must be extended to support the additional capabilities
10983 provided by IEEE Std 1003.1-2001.

10984 D.1.10 Command Language

10985 Users should be able to define procedures that combine simple tools and/or applications into
10986 higher-level components that perform to the specific needs of the user. The user should be able
10987 to store, recall, use, and modify these procedures. These procedures should employ a powerful
10988 command language that is used for recurring tasks in conforming applications (scripts) in the
10989 same way that it is used interactively to accomplish one-time tasks. The language and the
10990 utilities that it uses must be consistent between systems to reduce errors and retraining.

10991 D.1.11 Interactive Facilities

10992 Use the system to accomplish individual tasks at an interactive terminal. The interface should be
10993 consistent, intuitive, and offer usability enhancements to increase the productivity of terminal
10994 users, reduce errors, and minimize retraining costs. Online documentation or usage assistance
10995 should be available.

10996 D.1.12 Accomplish Multiple Tasks Simultaneously

10997 Access applications and interactive facilities from a single terminal without requiring serial
10998 execution: switch between multiple interactive tasks; schedule one-time or periodic background
10999 work; display the status of all work in progress or scheduled; influence the priority scheduling of
11000 work, when authorized.

11001 D.1.13 Complex Data Manipulation

11002 Manipulate data in files in complex ways: sort, merge, compare, translate, edit, format, pattern
11003 match, select subsets (strings, columns, fields, rows, and so on). These facilities should be
11004 available to both conforming applications and interactive users.

11005 D.1.14 File Hierarchy Manipulation

11006 Create, delete, move/rename, copy, backup/archive, and display files and directories. These
11007 facilities should be available to both conforming applications and interactive users.

11008 D.1.15 Locale Configuration

11009 Customize applications and interactive sessions for the cultural and language conventions of the
11010 user. Employ a wide variety of standard character encodings. These facilities should be available
11011 to both conforming applications and interactive users.

11012 D.1.16 Inter-User Communication

11013 Send messages or transfer files to other users on the same system or other systems on a network.
 11014 These facilities should be available to both conforming applications and interactive users.

11015 D.1.17 System Environment

11016 Display information about the status of the system (activities of users and their interactive and
 11017 background work, file system utilization, system time, configuration, and presence of optional
 11018 facilities) and the environment of the user (terminal characteristics, and so on). Inform the
 11019 system operator/administrator of problems. Control access to user files and other resources.

11020 D.1.18 Printing

11021 Output files on a variety of output device classes, accessing devices on local or network-
 11022 connected systems. Control (or influence) the formatting, priority scheduling, and output
 11023 distribution of work. These facilities should be available to both conforming applications and
 11024 interactive users.

11025 D.1.19 Software Development

11026 Develop (create and manage source files, compile/interpret, debug) portable open systems
 11027 applications and package them for distribution to, and updating of, other systems.

11028 D.2 Portability Capabilities

11029 This section describes the significant portability capabilities of IEEE Std 1003.1-2001 and
 11030 indicates how the user requirements listed in Section D.1 (on page 267) are addressed. The
 11031 capabilities are listed in the same format as the preceding user requirements; they are
 11032 summarized below:

- 11033 • Configuration Interrogation
- 11034 • Process Management
- 11035 • Access to Data
- 11036 • Access to the Environment
- 11037 • Access to Determinism and Performance Enhancements
- 11038 • Operating System-Dependent Profile
- 11039 • I/O Interaction
- 11040 • Internationalization Interaction
- 11041 • C-Language Extensions
- 11042 • Command Language
- 11043 • Interactive Facilities
- 11044 • Accomplish Multiple Tasks Simultaneously
- 11045 • Complex Data Manipulation
- 11046 • File Hierarchy Manipulation

- 11047 • Locale Configuration
- 11048 • Inter-User Communication
- 11049 • System Environment
- 11050 • Printing
- 11051 • Software Development

11052 D.2.1 Configuration Interrogation

11053 The *uname()* operation provides basic identification of the system. The *sysconf()*, *pathconf()*, and
 11054 *fpathconf()* functions and the *getconf* utility provide means to interrogate the implementation to
 11055 determine how to adapt to the environment in which it is running. These values can be either
 11056 static (indicating that all instances of the implementation have the same value) or dynamic
 11057 (indicating that different instances of the implementation have the different values, or that the
 11058 value may vary for other reasons, such as reconfiguration).

11059 Unsatisfied Requirements

11060 None directly. However, as new areas are added, there will be a need for additional capability in
 11061 this area.

11062 D.2.2 Process Management

11063 The *fork()*, *exec* family, *posix_spawn()*, and *posix_spawnnp()* functions provide for the creation of
 11064 new processes or the insertion of new applications into existing processes. The *_Exit()*, *_exit()*,
 11065 *exit()*, and *abort()* functions allow for the termination of a process by itself. The *wait()* and
 11066 *waitpid()* functions allow one process to deal with the termination of another.

11067 The *times()* function allows for basic measurement of times used by a process. Various
 11068 functions, including *fstat()*, *getegid()*, *geteuid()*, *getgid()*, *getgrgid()*, *getgrnam()*, *getlogin()*,
 11069 *getpid()*, *getppid()*, *getpwnam()*, *getpwuid()*, *getuid()*, *lstat()*, and *stat()*, provide for access to the
 11070 identifiers of processes and the identifiers and names of owners of processes (and files).

11071 The various functions operating on environment variables provide for communication of
 11072 information (primarily user-configurable defaults) from a parent to child processes.

11073 The operations on the current working directory control and interrogate the directory from
 11074 which relative filename searches start. The *umask()* function controls the default protections
 11075 applied to files created by the process.

11076 The *alarm()*, *pause()*, *sleep()*, *ualarm()*, and *usleep()* operations allow the process to suspend until
 11077 a timer has expired or to be notified when a period of time has elapsed. The *time()* operation
 11078 interrogates the current time and date.

11079 The signal mechanism provides for communication of events either from other processes or
 11080 from the environment to the application, and the means for the application to control the effect
 11081 of these events. The mechanism provides for external termination of a process and for a process
 11082 to suspend until an event occurs. The mechanism also provides for a value to be associated with
 11083 an event.

11084 Job control provides a means to group processes and control them as groups, and to control their
 11085 access to the function between the user and the system (the “controlling terminal”). It also
 11086 provides the means to suspend and resume processes.

11087 The Process Scheduling option provides control of the scheduling and priority of a process.

11088 The Message Passing option provides a means for interprocess communication involving small
11089 amounts of data.

11090 The Memory Management facilities provide control of memory resources and for the sharing of
11091 memory. This functionality is mandatory on XSI-conformant systems.

11092 The Threads facilities provide multiple flows of control with a process (threads),
11093 synchronization between threads, association of data with threads, and controlled cancellation
11094 of threads.

11095 The XSI interprocess communications functionality provide an alternate set of facilities to
11096 manipulate semaphores, message queues, and shared memory. These are provided on XSI-
11097 conformant systems to support conforming applications developed to run on UNIX systems.

11098 **D.2.3 Access to Data**

11099 The *open()*, *close()*, *fclose()*, *fopen()*, and *pipe()* functions provide for access to files and data.
11100 Such files may be regular files, interprocess data channels (pipes), or devices. Additional types
11101 of objects in the file system are permitted and are being contemplated for standardization.

11102 The *access()*, *chmod()*, *chown()*, *dup()*, *dup2()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lstat()*,
11103 *readlink()*, *realpath()*, *stat()*, and *utime()* functions allow for control and interrogation of file and
11104 file-related objects (including symbolic links), and their ownership, protections, and timestamps.

11105 The *fgetc()*, *fputc()*, *fread()*, *fseek()*, *fsetpos()*, *fwrite()*, *getc()*, *getchar()*, *lseek()*, *putchar()*, *putc()*,
11106 *read()*, and *write()* functions provide for data transfer from the application to files (in all their
11107 forms).

11108 The *closedir()*, *link()*, *mkdir()*, *opendir()*, *readdir()*, *rename()*, *rmdir()*, *rewinddir()*, and *unlink()*
11109 functions provide for a complete set of operations on directories. Directories can arbitrarily
11110 contain other directories, and a single file can be mentioned in more than one directory.

11111 The file-locking mechanism provides for advisory locking (protection during transactions) of
11112 ranges of bytes (in effect, records) in a file.

11113 The *confstr()*, *fpathconf()*, *pathconf()*, and *sysconf()* functions provide for enquiry as to the
11114 behavior of the system where variability is permitted.

11115 The Synchronized Input and Output option provides for assured commitment of data to media.

11116 The Asynchronous Input and Output option provides for initiation and control of asynchronous
11117 data transfers.

11118 **D.2.4 Access to the Environment**

11119 The operations and types in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11,
11120 General Terminal Interface are provided for access to asynchronous serial devices. The primary
11121 intended use for these is the controlling terminal for the application (the interaction point
11122 between the user and the system). They are general enough to be used to control any
11123 asynchronous serial device. The functions are also general enough to be used with many other
11124 device types as a user interface when some emulation is provided.

11125 Less detailed access is provided for other device types, but in many instances an application
11126 need not know whether an object in the file system is a device or a regular file to operate
11127 correctly.

11128 **Unsatisfied Requirements**

11129 Detailed control of common device classes, specifically magnetic tape, is not provided.

11130 **D.2.5 Bounded (Realtime) Response**

11131 The Realtime Signals Extension provides queued signals and the prioritization of the handling of
 11132 signals. The SCHED_FIFO, SCHED_SPORADIC, and SCHED_RR scheduling policies provide
 11133 control over processor allocation. The Semaphores option provides high-performance
 11134 synchronization. The Memory Management functions provide memory locking for control of
 11135 memory allocation, file mapping for high-performance, and shared memory for high-
 11136 performance interprocess communication. The Message Passing option provides for interprocess
 11137 communication without being dependent on shared memory.

11138 The Timers option provides a high resolution function called *nanosleep()* with a finer resolution
 11139 than the *sleep()* function.

11140 The Typed Memory Objects option, the Monotonic Clock option, and the Timeouts option
 11141 provide further facilities for applications to use to obtain predictable bounded response.

11142 **D.2.6 Operating System-Dependent Profile**

11143 IEEE Std 1003.1-2001 makes no distinction between text and binary files. The values of
 11144 EXIT_SUCCESS and EXIT_FAILURE are further defined.

11145 **Unsatisfied Requirements**

11146 None known, but the ISO C standard may contain some additional options that could be
 11147 specified.

11148 **D.2.7 I/O Interaction**

11149 IEEE Std 1003.1-2001 defines how each of the ISO C standard *stdio* functions interact with the
 11150 POSIX.1 operations, typically specifying the behavior in terms of POSIX.1 operations.

11151 **Unsatisfied Requirements**

11152 None.

11153 **D.2.8 Internationalization Interaction**

11154 The IEEE Std 1003.1-2001 environment operations provide a means to define the environment
 11155 for *setlocale()* and time functions such as *ctime()*. The *tzset()* function is provided to set time
 11156 conversion information.

11157 The *nl_langinfo()* function is provided as an XSI extension to query locale-specific cultural
 11158 settings.

11159 **Unsatisfied Requirements**

11160 None.

11161 D.2.9 C-Language Extensions

11162 The *setjmp()* and *longjmp()* functions are not defined to be cognizant of the signal masks defined
 11163 for POSIX.1. The *sigsetjmp()* and *siglongjmp()* functions are provided to fill this gap.

11164 The *_setjmp()* and *_longjmp()* functions are provided as XSI extensions to support historic
 11165 practice.

11166 Unsatisfied Requirements

11167 None.

11168 D.2.10 Command Language

11169 The shell command language, as described in the Shell and Utilities volume of
 11170 IEEE Std 1003.1-2001, Chapter 2, Shell Command Language, is a common language useful in
 11171 batch scripts, through an API to high-level languages (for the C-Language Binding option,
 11172 *system()* and *popen()*) and through an interactive terminal (see the *sh* utility). The shell language
 11173 has many of the characteristics of a high-level language, but it has been designed to be more
 11174 suitable for user terminal entry and includes interactive debugging facilities. Through the use of
 11175 pipelining, many complex commands can be constructed from combinations of data filters and
 11176 other common components. Shell scripts can be created, stored, recalled, and modified by the
 11177 user with simple editors.

11178 In addition to the basic shell language, the following utilities offer features that simplify and
 11179 enhance programmatic access to the utilities and provide features normally found only in high-
 11180 level languages: *basename*, *bc*, *command*, *dirname*, *echo*, *env*, *expr*, *false*, *printf*, *read*, *sleep*, *tee*, *test*,
 11181 *time**,² *true*, *wait*, *xargs*, and all of the special built-in utilities in the Shell and Utilities volume of
 11182 IEEE Std 1003.1-2001, Section 2.14, Special Built-In Utilities.

11183 Unsatisfied Requirements

11184 None.

11185 D.2.11 Interactive Facilities

11186 The utilities offer a common style of command-line interface through conformance to the Utility
 11187 Syntax Guidelines (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility
 11188 Syntax Guidelines) and the common utility defaults (see the Shell and Utilities volume of
 11189 IEEE Std 1003.1-2001, Section 1.11, Utility Description Defaults). The *sh* utility offers an
 11190 interactive command-line history and editing facility. The following utilities in the User
 11191 Portability Utilities option have been customized for interactive use: *alias*, *ex*, *fc*, *mailx*, *more*, *talk*,
 11192 *vi*, *unalias*, and *write*; the *man* utility offers online access to system documentation.

11193 _____

11194 2. The utilities listed with an asterisk here and later in this section are present only on systems which support the User Portability
 11195 Utilities option. There may be further restrictions on the utilities offered with various configuration option combinations; see the
 11196 individual utility descriptions.

11197 **Unsatisfied Requirements**

11198 The command line interface to individual utilities is as intuitive and consistent as historical
 11199 practice allows. Work underway based on graphical user interfaces may be more suitable for
 11200 novice or occasional users of the system.

11201 **D.2.12 Accomplish Multiple Tasks Simultaneously**

11202 The shell command language offers background processing through the asynchronous list
 11203 command form; see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.9, Shell
 11204 Commands. The *nohup* utility makes background processing more robust and usable. The *kill*
 11205 utility can terminate background jobs. When the User Portability Utilities option is supported,
 11206 the following utilities allow manipulation of jobs: *bg*, *fg*, and *jobs*. Also, if the User Portability
 11207 Utilities option is supported, the following can support periodic job scheduling, control, and
 11208 display: *at*, *batch*, *crontab*, *nice*, *ps*, and *renice*.

11209 **Unsatisfied Requirements**

11210 Terminals with multiple windows may be more suitable for some multi-tasking interactive uses
 11211 than the job control approach in IEEE Std 1003.1-2001. See the comments on graphical user
 11212 interfaces in Section D.2.11 (on page 274). The *nice* and *renice* utilities do not necessarily take
 11213 advantage of complex system scheduling algorithms that are supported by the realtime options
 11214 within IEEE Std 1003.1-2001.

11215 **D.2.13 Complex Data Manipulation**

11216 The following utilities address user requirements in this area: *asa*, *awk*, *bc*, *cmp*, *comm*, *csplit**, *cut*,
 11217 *dd*, *diff*, *ed*, *ex**, *expand**, *expr*, *find*, *fold*, *grep*, *head*, *join*, *od*, *paste*, *pr*, *printf*, *sed*, *sort*, *split**, *tabs**, *tail*,
 11218 *tr*, *unexpand**, *uniq*, *uudecode**, *uuencode**, and *wc*.

11219 **Unsatisfied Requirements**

11220 Sophisticated text formatting utilities, such as *troff* or *TeX*, are not included. Standards work in
 11221 the area of SGML may satisfy this.

11222 **D.2.14 File Hierarchy Manipulation**

11223 The following utilities address user requirements in this area: *basename*, *cd*, *chgrp*, *chmod*, *chown*,
 11224 *cksum*, *cp*, *dd*, *df**, *diff*, *dirname*, *du**, *find*, *ls*, *ln*, *mkdir*, *mkfifo*, *mv*, *patch**, *pathchk*, *pax*, *pwd*, *rm*, *rmdir*,
 11225 *test*, and *touch*.

11226 **Unsatisfied Requirements**

11227 Some graphical user interfaces offer more intuitive file manager components that allow file
 11228 manipulation through the use of icons for novice users.

11229 D.2.15 Locale Configuration

11230 The standard utilities are affected by the various *LC_* variables to achieve locale-dependent
 11231 operation: character classification, collation sequences, regular expressions and shell pattern
 11232 matching, date and time formats, numeric formatting, and monetary formatting. When the
 11233 POSIX2_LOCALEDEF option is supported, applications can provide their own locale definition
 11234 files. The following utilities address user requirements in this area: *date*, *ed*, *ex**, *find*, *grep*, *locale*,
 11235 *localedef*, *more**, *sed*, *sh*, *sort*, *tr*, *uniq*, and *vi**.

11236 The *iconv()*, *iconv_close()*, and *iconv_open()* functions are available to allow an application to
 11237 convert character data between supported character sets.

11238 The *gencat* utility and the *catopen()*, *catclose()*, and *catgets()* functions for message catalog
 11239 manipulation are available on XSI-conformant systems.

11240 Unsatisfied Requirements

11241 Some aspects of multi-byte character and state-encoded character encodings have not yet been
 11242 addressed. The C-language functions, such as *getopt()*, are generally limited to single-byte
 11243 characters. The effect of the *LC_MESSAGES* variable on message formats is only suggested at
 11244 this time.

11245 D.2.16 Inter-User Communication

11246 The following utilities address user requirements in this area: *cksum*, *mailx**, *mesg**, *patch**, *pax*,
 11247 *talk**, *uudecode**, *uuencode**, *who**, and *write**.

11248 The historical UUCP utilities are included on XSI-conformant systems.

11249 Unsatisfied Requirements

11250 None.

11251 D.2.17 System Environment

11252 The following utilities address user requirements in this area: *chgrp*, *chmod*, *chown*, *df**, *du**, *env*,
 11253 *getconf*, *id*, *logger*, *logname*, *mesg**, *newgrp**, *ps**, *stty*, *tput**, *tty*, *umask*, *uname*, and *who**.

11254 The *closelog()*, *openlog()*, *setlogmask()*, and *syslog()* functions provide System Logging facilities
 11255 on XSI-conformant systems; these are analogous to the *logger* utility.

11256 Unsatisfied Requirements

11257 None.

11258 D.2.18 Printing

11259 The following utilities address user requirements in this area: *pr* and *lp*.

11260 Unsatisfied Requirements

11261 There are no features to control the formatting or scheduling of the print jobs.

11262 D.2.19 Software Development

11263 The following utilities address user requirements in this area: *ar*, *asa*, *awk*, *c99*, *ctags**, *fort77*,
 11264 *getconf*, *getopts*, *lex*, *localedef*, *make*, *nm**, *od*, *patch**, *pax*, *strings**, *strip*, *time**, and *yacc*.

11265 The *system()*, *popen()*, *pclose()*, *regcomp()*, *regex()*, *regerror()*, *regfree()*, *fnmatch()*, *getopt()*,
 11266 *glob()*, *globfree()*, *wordexp()*, and *wordfree()* functions allow C-language programmers to access
 11267 some of the interfaces used by the utilities, such as argument processing, regular expressions,
 11268 and pattern matching.

11269 The SCCS source-code control system utilities are available on systems supporting the XSI
 11270 Development option.

11271 Unsatisfied Requirements

11272 There are no language-specific development tools related to languages other than C and
 11273 FORTRAN. The C tools are more complete and varied than the FORTRAN tools. There is no
 11274 data dictionary or other CASE-like development tools.

11275 D.2.20 Future Growth

11276 It is arguable whether or not all functionality to support applications is potentially within the
 11277 scope of IEEE Std 1003.1-2001. As a simple matter of practicality, it cannot be. Areas such as
 11278 graphics, application domain-specific functionality, windowing, and so on, should be in unique
 11279 standards. As such, they are properly “Unsatisfied Requirements” in terms of providing fully
 11280 conforming applications, but ones which are outside the scope of IEEE Std 1003.1-2001.

11281 However, as the standards evolve, certain functionality once considered “exotic” enough to be
 11282 part of a separate standard become common enough to be included in a core standard such as
 11283 this. Realtime and networking, for example, have both moved from separate standards (with
 11284 much difficult cross-referencing) into IEEE Std 1003.1 over time, and although no specific areas
 11285 have been identified for inclusion in future revisions, such inclusions seem likely.

11286 D.3 Profiling Considerations

11287 This section offers guidance to writers of profiles on how the configurable options, limits, and
 11288 optional behavior of IEEE Std 1003.1-2001 should be cited in profiles. Profile writers should
 11289 consult the general guidance in POSIX.0 when writing POSIX Standardized Profiles.

11290 The information in this section is an inclusive list of features that should be considered by profile
 11291 writers. Subsetting of IEEE Std 1003.1-2001 should follow the Base Definitions volume of
 11292 IEEE Std 1003.1-2001, Section 2.1.5.1, Subprofiling Considerations. A set of profiling options is
 11293 described in Appendix E (on page 291).

11294 D.3.1 Configuration Options

11295 There are two set of options suggested by IEEE Std 1003.1-2001: those for POSIX-conforming
 11296 systems and those for X/Open System Interface (XSI) conformance. The requirements for XSI
 11297 conformance are documented in the Base Definitions volume of IEEE Std 1003.1-2001 and not
 11298 discussed further here, as they superset the POSIX conformance requirements.

11299 D.3.2 Configuration Options (Shell and Utilities)

11300 There are three broad optional configurations for the Shell and Utilities volume of
 11301 IEEE Std 1003.1-2001: basic execution system, development system, and user portability
 11302 interactive system. The options to support these, and other minor configuration options, are
 11303 listed in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance. Profile
 11304 writers should consult the following list and the comments concerning user requirements
 11305 addressed by various components in Section D.2 (on page 270).

11306 POSIX2_UPE

11307 The system supports the User Portability Utilities option.

11308 This option is a requirement for a user portability interactive system. It is required
 11309 frequently except for those systems, such as embedded realtime or dedicated application
 11310 systems, that support little or no interactive time-sharing work by users or operators. XSI-
 11311 conformant systems support this option.

11312 POSIX2_SW_DEV

11313 The system supports the Software Development Utilities option.

11314 This option is required by many systems, even those in which actual software development
 11315 does not occur. The *make* utility, in particular, is required by many application software
 11316 packages as they are installed onto the system. If POSIX2_C_DEV is supported,
 11317 POSIX2_SW_DEV is almost a mandatory requirement because of *ar* and *make*.

11318 POSIX2_C_BIND

11319 The system supports the C-Language Bindings option.

11320 This option is required on some implementations developing complex C applications or on
 11321 any system installing C applications in source form that require the functions in this option.
 11322 The *system()* and *popen()* functions, in particular, are widely used by applications; the
 11323 others are rather more specialized.

11324 POSIX2_C_DEV

11325 The system supports the C-Language Development Utilities option.

11326 This option is required by many systems, even those in which actual C-language software
 11327 development does not occur. The *c99* utility, in particular, is required by many application
 11328 software packages as they are installed onto the system. The *lex* and *yacc* utilities are used
 11329 less frequently.

11330 POSIX2_FORT_DEV

11331 The system supports the FORTRAN Development Utilities option

11332 As with C, this option is needed on any system developing or installing FORTRAN
 11333 applications in source form.

11334 POSIX2_FORT_RUN

11335 The system supports the FORTRAN Runtime Utilities option.

11336 This option is required for some FORTRAN applications that need the *asa* utility to convert
 11337 Hollerith printing statement output. It is unknown how frequently this occurs.

11338 POSIX2_LOCALEDEF

11339 The system supports the creation of locales.

11340 This option is needed if applications require their own customized locale definitions to
 11341 operate. It is presently unknown whether many applications are dependent on this.
 11342 However, the option is virtually mandatory for systems in which internationalized
 11343 applications are developed.

11344	XSI-conformant systems support this option.	
11345	POSIX2_PBS	
11346	The system supports the Batch Environment Services and Utilities option.	
11347	POSIX2_PBS_ACCOUNTING	
11348	The system supports the optional feature of accounting within the Batch Environment	
11349	Services and Utilities option. It will be required in servers that implement the optional	
11350	feature of accounting.	
11351	POSIX2_PBS_CHECKPOINT	
11352	The system supports the optional feature of checkpoint/restart within the Batch	
11353	Environment Services and Utilities option.	
11354	POSIX2_PBS_LOCATE	
11355	The system supports the optional feature of locating batch jobs within the Batch	
11356	Environment Services and Utilities option.	
11357	POSIX2_PBS_MESSAGE	
11358	The system supports the optional feature of sending messages to batch jobs within the	
11359	Batch Environment Services and Utilities option.	
11360	POSIX2_PBS_TRACK	
11361	The system supports the optional feature of tracking batch jobs within the Batch	
11362	Environment Services and Utilities option.	
11363	POSIX2_CHAR_TERM	
11364	The system supports at least one terminal type capable of all operations described in	
11365	IEEE Std 1003.1-2001.	
11366	On systems with POSIX2_UPE, this option is almost always required. It was developed	
11367	solely to allow certain specialized vendors and user applications to bypass the requirement	
11368	for general-purpose asynchronous terminal support. For example, an application and	
11369	system that was suitable for block-mode terminals, such as IBM 3270s, would not need this	
11370	option.	
11371	XSI-conformant systems support this option.	

11372 **D.3.3 Configurable Limits**

11373	Very few of the limits need to be increased for profiles. No profile can cite lower values.	
11374	{POSIX2_BC_BASE_MAX}	
11375	{POSIX2_BC_DIM_MAX}	
11376	{POSIX2_BC_SCALE_MAX}	
11377	{POSIX2_BC_STRING_MAX}	
11378	No increase is anticipated for any of these <i>bc</i> values, except for very specialized applications	
11379	involving huge numbers.	
11380	{POSIX2_COLL_WEIGHTS_MAX}	
11381	Some natural languages with complex collation requirements require an increase from the	
11382	default 2 to 4; no higher numbers are anticipated.	
11383	{POSIX2_EXPR_NEST_MAX}	
11384	No increase is anticipated.	
11385	{POSIX2_LINE_MAX}	
11386	This number is much larger than most historical applications have been able to use. At some	
11387	future time, applications may be rewritten to take advantage of even larger values.	

11388 {POSIX2_RE_DUP_MAX}
 11389 No increase is anticipated.

11390 {POSIX2_VERSION}
 11391 This is actually not a limit, but a standard version stamp. Generally, a profile should specify
 11392 the Shell and Utilities volume of IEEE Std 1003.1-2001, Chapter 2, Shell Command Language
 11393 by name in the normative references section, not this value.

11394 **D.3.4 Configuration Options (System Interfaces)**

11395 {NGROUPS_MAX}
 11396 A non-zero value indicates that the implementation supports supplementary groups.

11397 This option is needed where there is a large amount of shared use of files, but where a
 11398 certain amount of protection is needed. Many profiles³ are known to require this option; it
 11399 should only be required if needed, but it should never be prohibited.

11400 _POSIX_ADVISORY_INFO
 11401 The system provides advisory information for file management.

11402 This option allows the application to specify advisory information that can be used to
 11403 achieve better or even deterministic response time in file manager or input and output
 11404 operations.

11405 _POSIX_ASYNCHRONOUS_IO
 11406 The system provides concurrent process execution and input and output transfers.

11407 This option was created to support historical systems that did not provide the feature. It
 11408 should only be required if needed, but it should never be prohibited.

11409 _POSIX_BARRIERS
 11410 The system supports barrier synchronization.

11411 This option was created to allow efficient synchronization of multiple parallel threads in
 11412 multi-processor systems in which the operation is supported in part by the hardware
 11413 architecture.

11414 _POSIX_CHOWN_RESTRICTED
 11415 The system restricts the right to “give away” files to other users.

11416 This option should be carefully investigated before it is required. Some applications expect
 11417 that they can change the ownership of files in this way. It is provided where either security
 11418 or system account requirements cause this ability to be a problem. It is also known to be
 11419 specified in many profiles.

11420 _POSIX_CLOCK_SELECTION
 11421 The system supports the Clock Selection option.

11422 This option allows applications to request a high resolution sleep in order to suspend a
 11423 thread during a relative time interval, or until an absolute time value, using the desired
 11424 clock. It also allows the application to select the clock used in a *pthread_cond_timedwait()*
 11425 function call.

11426 _____

11427 3. There are no formally approved profiles of IEEE Std 1003.1-2001 at the time of publication; the reference here is to various
 11428 profiles generated by private bodies or governments.

11429 `_POSIX_CPUTIME`

11430 The system supports the Process CPU-Time Clocks option.

11431 This option allows applications to use a new clock that measures the execution times of
 11432 processes or threads, and the possibility to create timers based upon these clocks, for
 11433 runtime detection (and treatment) of execution time overruns.

11434 `_POSIX_FSYNC`

11435 The system supports file synchronization requests.

11436 This option was created to support historical systems that did not provide the feature.
 11437 Applications that are expecting guaranteed completion of their input and output operations
 11438 should require the `_POSIX_SYNC_IO` option. This option should never be prohibited.

11439 XSI-conformant systems support this option.

11440 `_POSIX_IPV6`

11441 The system supports facilities related to Internet Protocol Version 6 (IPv6).

11442 This option was created to allow systems to transition to IPv6.

11443 `_POSIX_JOB_CONTROL`

11444 Job control facilities are mandatory in IEEE Std 1003.1-2001.

11445 The option was created primarily to support historical systems that did not provide the
 11446 feature. Many existing profiles now require it; it should only be required if needed, but it
 11447 should never be prohibited. Most applications that use it can run when it is not present,
 11448 although with a degraded level of user convenience.

11449 `_POSIX_MAPPED_FILES`

11450 The system supports the mapping of regular files into the process address space.

11451 XSI-conformant systems support this option.

11452 Both this option and the Shared Memory Objects option provide shared access to memory
 11453 objects in the process address space. The functions defined under this option provide the
 11454 functionality of existing practice for mapping regular files. This functionality was deemed
 11455 unnecessary, if not inappropriate, for embedded systems applications and, hence, is
 11456 provided under this option. It should only be required if needed, but it should never be
 11457 prohibited.

11458 `_POSIX_MEMLOCK`

11459 The system supports the locking of the address space.

11460 This option was created to support historical systems that did not provide the feature. It
 11461 should only be required if needed, but it should never be prohibited.

11462 `_POSIX_MEMLOCK_RANGE`

11463 The system supports the locking of specific ranges of the address space.

11464 For applications that have well-defined sections that need to be locked and others that do
 11465 not, IEEE Std 1003.1-2001 supports an optional set of functions to lock or unlock a range of
 11466 process addresses. The following are two reasons for having a means to lock down a
 11467 specific range:

- 11468 1. An asynchronous event handler function that must respond to external events in a
 11469 deterministic manner such that page faults cannot be tolerated
- 11470 2. An input/output “buffer” area that is the target for direct-to-process I/O, and the
 11471 overhead of implicit locking and unlocking for each I/O call cannot be tolerated

- 11472 It should only be required if needed, but it should never be prohibited.
- 11473 _posix_memory_protection
- 11474 The system supports memory protection.
- 11475 XSI-conformant systems support this option.
- 11476 The provision of this option typically imposes additional hardware requirements. It should
- 11477 never be prohibited.
- 11478 _posix_prioritized_io
- 11479 The system provides prioritization for input and output operations.
- 11480 The use of this option may interfere with the ability of the system to optimize input and
- 11481 output throughput. It should only be required if needed, but it should never be prohibited.
- 11482 _posix_message_passing
- 11483 The system supports the passing of messages between processes.
- 11484 This option was created to support historical systems that did not provide the feature. The
- 11485 functionality adds a high-performance XSI interprocess communication facility for local
- 11486 communication. It should only be required if needed, but it should never be prohibited.
- 11487 _posix_monotonic_clock
- 11488 The system supports the Monotonic Clock option.
- 11489 This option allows realtime applications to rely on a monotonically increasing clock that
- 11490 does not jump backwards, and whose value does not change except for the regular ticking
- 11491 of the clock.
- 11492 _posix_priority_scheduling
- 11493 The system provides priority-based process scheduling.
- 11494 Support of this option provides predictable scheduling behavior, allowing applications to
- 11495 determine the order in which processes that are ready to run are granted access to a
- 11496 processor. It should only be required if needed, but it should never be prohibited.
- 11497 _posix_realtime_signals
- 11498 The system provides prioritized, queued signals with associated data values.
- 11499 This option was created to support historical systems that did not provide the features. It
- 11500 should only be required if needed, but it should never be prohibited.
- 11501 _posix_regexp
- 11502 Support for regular expression facilities is mandatory in IEEE Std 1003.1-2001.
- 11503 _posix_saved_ids
- 11504 Support for this feature is mandatory in IEEE Std 1003.1-2001.
- 11505 Certain classes of applications rely on it for proper operation, and there is no alternative
- 11506 short of giving the application root privileges on most implementations that did not provide
- 11507 _posix_saved_ids.
- 11508 _posix_semaphores
- 11509 The system provides counting semaphores.
- 11510 This option was created to support historical systems that did not provide the feature. It
- 11511 should only be required if needed, but it should never be prohibited.
- 11512 _posix_shared_memory_objects
- 11513 The system supports the mapping of shared memory objects into the process address space.

- 11514 Both this option and the Memory Mapped Files option provide shared access to memory
 11515 objects in the process address space. The functions defined under this option provide the
 11516 functionality of existing practice for shared memory objects. This functionality was deemed
 11517 appropriate for embedded systems applications and, hence, is provided under this option. It
 11518 should only be required if needed, but it should never be prohibited.
- 11519 `_POSIX_SHELL`
 11520 Support for the *sh* utility command line interpreter is mandatory in IEEE Std 1003.1-2001.
- 11521 `_POSIX_SPAWN`
 11522 The system supports the spawn option.
- 11523 This option provides applications with an efficient mechanism to spawn execution of a new
 11524 process.
- 11525 `_POSIX_SPINLOCKS`
 11526 The system supports spin locks.
- 11527 This option was created to support a simple and efficient synchronization mechanism for
 11528 threads executing in multi-processor systems.
- 11529 `_POSIX_SPORADIC_SERVER`
 11530 The system supports the sporadic server scheduling policy.
- 11531 This option provides applications with a new scheduling policy for scheduling aperiodic
 11532 processes or threads in hard realtime applications.
- 11533 `_POSIX_SYNCHRONIZED_IO`
 11534 The system supports guaranteed file synchronization.
- 11535 This option was created to support historical systems that did not provide the feature.
 11536 Applications that are expecting guaranteed completion of their input and output operations
 11537 should require this option, rather than the File Synchronization option. It should only be
 11538 required if needed, but it should never be prohibited.
- 11539 `_POSIX_THREADS`
 11540 The system supports multiple threads of control within a single process.
- 11541 This option was created to support historical systems that did not provide the feature.
 11542 Applications written assuming a multi-threaded environment would be expected to require
 11543 this option. It should only be required if needed, but it should never be prohibited.
- 11544 XSI-conformant systems support this option.
- 11545 `_POSIX_THREAD_ATTR_STACKADDR`
 11546 The system supports specification of the stack address for a created thread.
- 11547 Applications may take advantage of support of this option for performance benefits, but
 11548 dependence on this feature should be minimized. This option should never be prohibited.
- 11549 XSI-conformant systems support this option.
- 11550 `_POSIX_THREAD_ATTR_STACKSIZE`
 11551 The system supports specification of the stack size for a created thread.
- 11552 Applications may require this option in order to ensure proper execution, but such usage
 11553 limits portability and dependence on this feature should be minimized. It should only be
 11554 required if needed, but it should never be prohibited.
- 11555 XSI-conformant systems support this option.

11556 `_POSIX_THREAD_PRIORITY_SCHEDULING`
11557 The system provides priority-based thread scheduling.

11558 Support of this option provides predictable scheduling behavior, allowing applications to
11559 determine the order in which threads that are ready to run are granted access to a processor.
11560 It should only be required if needed, but it should never be prohibited.

11561 `_POSIX_THREAD_PRIO_INHERIT`
11562 The system provides mutual-exclusion operations with priority inheritance.

11563 Support of this option provides predictable scheduling behavior, allowing applications to
11564 determine the order in which threads that are ready to run are granted access to a processor.
11565 It should only be required if needed, but it should never be prohibited.

11566 `_POSIX_THREAD_PRIO_PROTECT`
11567 The system supports a priority ceiling emulation protocol for mutual-exclusion operations.

11568 Support of this option provides predictable scheduling behavior, allowing applications to
11569 determine the order in which threads that are ready to run are granted access to a processor.
11570 It should only be required if needed, but it should never be prohibited.

11571 `_POSIX_THREAD_PROCESS_SHARED`
11572 The system provides shared access among multiple processes to synchronization objects.

11573 This option was created to support historical systems that did not provide the feature. It
11574 should only be required if needed, but it should never be prohibited.

11575 XSI-conformant systems support this option.

11576 `_POSIX_THREAD_SAFE_FUNCTIONS`
11577 The system provides thread-safe versions of all of the POSIX.1 functions.

11578 This option is required if the Threads option is supported. This is a separate option because
11579 thread-safe functions are useful in implementations providing other mechanisms for
11580 concurrency. It should only be required if needed, but it should never be prohibited.

11581 XSI-conformant systems support this option.

11582 `_POSIX_THREAD_SPORADIC_SERVER`
11583 The system supports the thread sporadic server scheduling policy.

11584 Support for this option provides applications with a new scheduling policy for scheduling
11585 aperiodic threads in hard realtime applications.

11586 `_POSIX_TIMEOUTS`
11587 The system provides timeouts for some blocking services.

11588 This option was created to provide a timeout capability to system services, thus allowing
11589 applications to include better error detection, and recovery capabilities.

11590 `_POSIX_TIMERS`
11591 The system provides higher resolution clocks with multiple timers per process.

11592 This option was created to support historical systems that did not provide the features. This
11593 option is appropriate for applications requiring higher resolution timestamps or needing to
11594 control the timing of multiple activities. It should only be required if needed, but it should
11595 never be prohibited.

11596 `_POSIX_TRACE`
11597 The system supports the Trace option.

11598 This option was created to allow applications to perform tracing.

11599 _posix_trace_event_filter

11600 The system supports the Trace Event Filter option.

11601 This option is dependent on support of the Trace option.

11602 _posix_trace_inherit

11603 The system supports the Trace Inherit option.

11604 This option is dependent on support of the Trace option.

11605 _posix_trace_log

11606 The system supports the Trace Log option.

11607 This option is dependent on support of the Trace option.

11608 _posix_typed_memory_objects

11609 The system supports the Typed Memory Objects option.

11610 This option was created to allow realtime applications to access different kinds of physical

11611 memory, and allow processes in these applications to share portions of this memory.

11612 **D.3.5 Configurable Limits**

11613 In general, the configurable limits in the **<limits.h>** header defined in the Base Definitions

11614 volume of IEEE Std 1003.1-2001 have been set to minimal values; many applications or

11615 implementations may require larger values. No profile can cite lower values.

11616 {AIO_LISTIO_MAX}

11617 The current minimum is likely to be inadequate for most applications. It is expected that

11618 this value will be increased by profiles requiring support for list input and output

11619 operations.

11620 {AIO_MAX}

11621 The current minimum is likely to be inadequate for most applications. It is expected that

11622 this value will be increased by profiles requiring support for asynchronous input and

11623 output operations.

11624 {AIO_PRIO_DELTA_MAX}

11625 The functionality associated with this limit is needed only by sophisticated applications. It

11626 is not expected that this limit would need to be increased under a general-purpose profile.

11627 {ARG_MAX}

11628 The current minimum is likely to need to be increased for profiles, particularly as larger

11629 amounts of information are passed through the environment. Many implementations are

11630 believed to support larger values.

11631 {CHILD_MAX}

11632 The current minimum is suitable only for systems where a single user is not running

11633 applications in parallel. It is significantly too low for any system also requiring windows,

11634 and if _POSIX_JOB_CONTROL is specified, it should be raised.

11635 {CLOCKRES_MIN}

11636 It is expected that profiles will require a finer granularity clock, perhaps as fine as 1 μ s,

11637 represented by a value of 1 000 for this limit.

11638 {DELAYTIMER_MAX}

11639 It is believed that most implementations will provide larger values.

11640	{LINK_MAX}
11641	For most applications and usage, the current minimum is adequate. Many implementations
11642	have a much larger value, but this should not be used as a basis for raising the value unless
11643	the applications to be used require it.
11644	{LOGIN_NAME_MAX}
11645	This is not actually a limit, but an implementation parameter. No profile should impose a
11646	requirement on this value.
11647	{MAX_CANON}
11648	For most purposes, the current minimum is adequate. Unless high-speed burst serial
11649	devices are used, it should be left as is.
11650	{MAX_INPUT}
11651	See {MAX_CANON}.
11652	{MQ_OPEN_MAX}
11653	The current minimum should be adequate for most profiles.
11654	{MQ_PRIO_MAX}
11655	The current minimum corresponds to the required number of process scheduling priorities.
11656	Many realtime practitioners believe that the number of message priority levels ought to be
11657	the same as the number of execution scheduling priorities.
11658	{NAME_MAX}
11659	Many implementations now support larger values, and many applications and users
11660	assume that larger names can be used. Many existing profiles also specify a larger value.
11661	Specifying this value will reduce the number of conforming implementations, although this
11662	might not be a significant consideration over time. Values greater than 255 should not be
11663	required.
11664	{NGROUPS_MAX}
11665	The value selected will typically be 8 or larger.
11666	{OPEN_MAX}
11667	The historically common value for this has been 20. Many implementations support larger
11668	values. If applications that use larger values are anticipated, an appropriate value should be
11669	specified.
11670	{PAGESIZE}
11671	This is not actually a limit, but an implementation parameter. No profile should impose a
11672	requirement on this value.
11673	{PATH_MAX}
11674	Historically, the minimum has been either 1024 or indefinite, depending on the
11675	implementation. Few applications actually require values larger than 256, but some users
11676	may create file hierarchies that must be accessed with longer paths. This value should only
11677	be changed if there is a clear requirement.
11678	{PIPE_BUF}
11679	The current minimum is adequate for most applications. Historically, it has been larger. If
11680	applications that write single transactions larger than this are anticipated, it should be
11681	increased. Applications that write lines of text larger than this probably do not need it
11682	increased, as the text line is delimited by a <newline>.
11683	{POSIX_VERSION}
11684	This is actually not a limit, but a standard version stamp. Generally, a profile should specify
11685	IEEE Std 1003.1-2001 by a name in the normative references section, not this value.

11686	{PTHREAD_DESTRUCTOR_ITERATIONS}
11687	It is unlikely that applications will need larger values to avoid loss of memory resources.
11688	{PTHREAD_KEYS_MAX}
11689	The current value should be adequate for most profiles.
11690	{PTHREAD_STACK_MIN}
11691	This should not be treated as an actual limit, but as an implementation parameter. No
11692	profile should impose a requirement on this value.
11693	{PTHREAD_THREADS_MAX}
11694	It is believed that most implementations will provide larger values.
11695	{RTSIG_MAX}
11696	The current limit was chosen so that the set of POSIX.1 signal numbers can fit within a 32-
11697	bit field. It is recognized that most existing implementations define many more signals than
11698	are specified in POSIX.1 and, in fact, many implementations have already exceeded 32
11699	signals (including the “null signal”). Support of {_POSIX_RTSIG_MAX} additional signals
11700	may push some implementations over the single 32-bit word line, but is unlikely to push
11701	any implementations that are already over that line beyond the 64 signal line.
11702	{SEM_NSEMS_MAX}
11703	The current value should be adequate for most profiles.
11704	{SEM_VALUE_MAX}
11705	The current value should be adequate for most profiles.
11706	{SSIZE_MAX}
11707	This limit reflects fundamental hardware characteristics (the size of an integer), and should
11708	not be specified unless it is clearly required. Extreme care should be taken to assure that
11709	any value that might be specified does not unnecessarily eliminate implementations
11710	because of accidents of hardware design.
11711	{STREAM_MAX}
11712	This limit is very closely related to {OPEN_MAX}. It should never be larger than
11713	{OPEN_MAX}, but could reasonably be smaller for application areas where most files are
11714	not accessed through <i>stdio</i> . Some implementations may limit {STREAM_MAX} to 20 but
11715	allow {OPEN_MAX} to be considerably larger. Such implementations should be allowed for
11716	if the applications permit.
11717	{TIMER_MAX}
11718	The current limit should be adequate for most profiles, but it may need to be larger for
11719	applications with a large number of asynchronous operations.
11720	{TTY_NAME_MAX}
11721	This is not actually a limit, but an implementation parameter. No profile should impose a
11722	requirement on this value.
11723	{TZNAME_MAX}
11724	The minimum has been historically adequate, but if longer timezone names are anticipated
11725	(particularly such values as UTC−1), this should be increased.

11726 D.3.6 Optional Behavior

11727 In IEEE Std 1003.1-2001, there are no instances of the terms unspecified, undefined,
11728 implementation-defined, or with the verbs “may” or “need not”, that the developers of
11729 IEEE Std 1003.1-2001 anticipate or sanction as suitable for profile or test method citation. All of
11730 these are merely warnings to conforming applications to avoid certain areas that can vary from
11731 system to system, and even over time on the same system. In many cases, these terms are used
11732 explicitly to support extensions, but profiles should not anticipate and require such extensions;
11733 future versions of IEEE Std 1003.1 may do so.

11734 / *Rationale (Informative)*

11735 **Part E:**

11736 **Subprofiling Considerations**

11737 *The Open Group*

11738 *The Institute of Electrical and Electronics Engineers, Inc.*

Subprofiling Considerations (Informative)

This section contains further information to satisfy the requirement that the project scope enable subprofiling of IEEE Std 1003.1-2001. The original intent was to have included a set of options similar to the “Units of Functionality” contained in IEEE Std 1003.13-1998. However, as the development of IEEE Std 1003.1-2001 continued, the standard developers felt it premature to fix these in normative text. The approach instead has been to include a general requirement in normative text regarding subprofiling and to include an informative section (here) containing a proposed set of subprofiling options.

E.1 Subprofiling Option Groups

The following Option Groups⁴ are defined to support profiling. Systems claiming support to IEEE Std 1003.1-2001 need not implement these options apart from the requirements stated in the Base Definitions volume of IEEE Std 1003.1-2001, Section 2.1.3, POSIX Conformance. These Option Groups allow profiles to subset the System Interfaces volume of IEEE Std 1003.1-2001 by collecting sets of related functions.

POSIX_C_LANG_JUMP: Jump Functions

longjmp(), *setjmp()*

POSIX_C_LANG_MATH: Maths Library

acos(), *acosf()*, *acosh()*, *acoshf()*, *acoshl()*, *acosl()*, *asin()*, *asinf()*, *asinh()*, *asinhf()*, *asinhf()*, *asinhl()*, *asinl()*, *atan()*, *atan2()*, *atan2f()*, *atan2l()*, *atanf()*, *atanh()*, *atanhf()*, *atanhl()*, *atanl()*, *cabs()*, *cabsf()*, *cabsl()*, *cacos()*, *cacosf()*, *cacosh()*, *cacoshf()*, *cacoshl()*, *cacosl()*, *carg()*, *cargf()*, *cargl()*, *casin()*, *casinf()*, *casinh()*, *casinhf()*, *casinhf()*, *casinl()*, *catan()*, *catanf()*, *catanh()*, *catanhf()*, *catanhf()*, *catanhl()*, *catanl()*, *cbrt()*, *cbrtf()*, *cbrtl()*, *ccos()*, *ccosf()*, *ccosh()*, *ccoshf()*, *ccoshl()*, *ccosl()*, *ceil()*, *ceilf()*, *ceilf()*, *cexp()*, *cexpf()*, *cexpl()*, *cimag()*, *cimagf()*, *cimagl()*, *clog()*, *clogf()*, *clogl()*, *conj()*, *conjf()*, *conjl()*, *copysign()*, *copysignf()*, *copysignl()*, *cos()*, *cosf()*, *cosh()*, *coshf()*, *coshf()*, *cosl()*, *cpow()*, *cpowf()*, *cpowl()*, *cproj()*, *cprojf()*, *cprojl()*, *creal()*, *crealf()*, *creall()*, *csin()*, *csinf()*, *csinh()*, *csinhf()*, *csinhf()*, *csinl()*, *csqrt()*, *csqrtf()*, *csqrtl()*, *ctan()*, *ctanf()*, *ctanh()*, *ctanhf()*, *ctanhf()*, *ctanl()*, *erf()*, *erfc()*, *erfcf()*, *erfcl()*, *erff()*, *erfl()*, *exp()*, *exp2()*, *exp2f()*, *exp2l()*, *expf()*, *expl()*, *expm1()*, *expm1f()*, *expm1l()*, *fabs()*, *fabsf()*, *fabsf()*, *fdim()*, *fdimf()*, *fdiml()*, *floor()*, *floorf()*, *floorl()*, *fma()*, *fmaf()*, *fmaf()*, *fmax()*, *fmaxf()*, *fmaxl()*, *fmin()*, *fminf()*, *fminl()*, *fmod()*, *fmodf()*, *fmodl()*, *fpclassify()*, *frexp()*, *frexpf()*, *frexpl()*, *hypot()*, *hypotf()*, *hypotl()*, *ilogb()*, *ilogbf()*, *ilogbl()*, *isfinite()*, *isgreater()*, *isgreaterequal()*, *isinf()*, *isless()*, *islessequal()*, *islessgreater()*, *isnan()*, *isnormal()*, *isunordered()*, *ldexp()*, *ldexpf()*, *ldexpl()*, *lgamma()*, *lgammaf()*, *lgammal()*, *llrint()*, *llrintf()*, *llrintl()*, *llround()*, *llroundf()*, *llroundl()*, *log()*, *log10()*, *log10f()*, *log10l()*, *log1p()*, *log1pf()*, *log1pl()*, *log2()*, *log2f()*, *log2l()*, *logb()*, *logbf()*, *logbl()*, *logf()*, *logl()*, *lrint()*, *lrintf()*, *lrintl()*, *lround()*, *lroundf()*, *lroundl()*, *modf()*, *modff()*, *modfl()*, *nan()*, *nanf()*, *nanl()*, *nearbyint()*, *nearbyintf()*, *nearbyintl()*, *nextafter()*, *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, *nexttowardl()*, *pow()*, *powf()*, *powf()*, *powl()*, *remainder()*, *remainderf()*, *remainderl()*, *remquo()*, *remquoof()*, *remquoof()*, *rint()*, *rintf()*, *rintl()*, *round()*, *roundf()*, *roundl()*, *scalbln()*, *scalblnf()*, *scalblnl()*, *scalbn()*, *scalbnf()*, *scalbnl()*, *signbit()*, *sin()*, *sinf()*, *sinh()*, *sinhf()*, *sinhl()*, *sinl()*, *sqrt()*, *sqrtf()*, *sqrtl()*, *tan()*, *tanf()*,

4. These are equivalent to the Units of Functionality from IEEE Std 1003.13-1998.

11782 *tanh()*, *tanhf()*, *tanhf()*, *tanhl()*, *tanl()*, *tgamma()*, *tgammaf()*, *tgammaf()*, *trunc()*, *truncf()*, *truncf()*

11783 POSIX_C_LANG_SUPPORT: General ISO C Library

11784 *abs()*, *asctime()*, *atof()*, *atoi()*, *atol()*, *atoll()*, *bsearch()*, *calloc()*, *ctime()*, *difftime()*, *div()*,
 11785 *feclearexcept()*, *fegetenv()*, *fegetexceptflag()*, *fegetround()*, *fehldexcept()*, *feraiseexcept()*,
 11786 *fesetenv()*, *fesetexceptflag()*, *fesetround()*, *fetestexcept()*, *feupdateenv()*, *free()*, *gmtime()*,
 11787 *imaxabs()*, *imaxdiv()*, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*,
 11788 *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *labs()*, *ldiv()*, *llabs()*, *lldiv()*, *localeconv()*,
 11789 *localtime()*, *malloc()*, *memchr()*, *memcmp()*, *memcpy()*, *memmove()*, *memset()*, *mktime()*,
 11790 *qsort()*, *rand()*, *realloc()*, *setlocale()*, *snprintf()*, *sprintf()*, *srand()*, *sscanf()*, *strcat()*, *strchr()*,
 11791 *strcmp()*, *strcoll()*, *strcpy()*, *strcspn()*, *strerror()*, *strftime()*, *strlen()*, *strncat()*, *strncpy()*,
 11792 *strncpy()*, *strpbrk()*, *strrchr()*, *strspn()*, *strstr()*, *strtod()*, *strtof()*, *strtoimax()*, *strtok()*, *strtol()*,
 11793 *strtold()*, *strtoll()*, *strtoul()*, *strtoull()*, *strtoimax()*, *strxfrm()*, *time()*, *tolower()*, *toupper()*,
 11794 *tzname()*, *tzset()*, *va_arg()*, *va_copy()*, *va_end()*, *va_start()*, *vsnprintf()*, *vsprintf()*, *vsscanf()*

11795 POSIX_C_LANG_SUPPORT_R: Thread-Safe General ISO C Library

11796 *asctime_r()*, *ctime_r()*, *gmtime_r()*, *localtime_r()*, *rand_r()*, *strerror_r()*, *strtok_r()*

11797 POSIX_C_LANG_WIDE_CHAR: Wide-Character ISO C Library

11798 *btowc()*, *iswalnum()*, *iswalnum()*, *iswalpha()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*,
 11799 *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *mblen()*, *mbrlen()*,
 11800 *mbrtowc()*, *mbsinit()*, *mbsrtowcs()*, *mbstowcs()*, *mbtowc()*, *swprintf()*, *swscanf()*, *towctrans()*,
 11801 *towlower()*, *towupper()*, *vswprintf()*, *vswscanf()*, *wcrtomb()*, *wcscat()*, *wcschr()*, *wcscmp()*,
 11802 *wscoll()*, *wscpy()*, *wcscspn()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*,
 11803 *wcspbrk()*, *wcsrchr()*, *wcstombs()*, *wcsspn()*, *wcsstr()*, *wcstod()*, *wcstof()*, *wcstoimax()*,
 11804 *wcstok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstombs()*, *wcstoul()*, *wcstoull()*, *wcstoumax()*,
 11805 *wcsxfrm()*, *wctob()*, *wctomb()*, *wctrans()*, *wctype()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*,
 11806 *wmemmove()*, *wmemset()*

11807 POSIX_C_LIB_EXT: General C Library Extension

11808 *fnmatch()*, *getopt()*, *optarg*, *opterr*, *optind*, *optopt*

11809 POSIX_DEVICE_IO: Device Input and Output

11810 *FD_CLR()*, *FD_ISSET()*, *FD_SET()*, *FD_ZERO()*, *clearerr()*, *close()*, *fclose()*, *fdopen()*, *feof()*,
 11811 *ferror()*, *fflush()*, *fgetc()*, *fgets()*, *fileno()*, *fopen()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *freopen()*,
 11812 *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *open()*, *perror()*, *printf()*, *pselect()*, *putc()*, *putchar()*,
 11813 *puts()*, *read()*, *scanf()*, *select()*, *setbuf()*, *setvbuf()*, *stderr*, *stdin*, *stdout*, *ungetc()*, *vfprintf()*,
 11814 *vfsprintf()*, *vprintf()*, *vscanf()*, *write()*

11815 POSIX_DEVICE_SPECIFIC: General Terminal

11816 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *ctermid()*, *isatty()*, *tcdrain()*, *tcflow()*,
 11817 *tcflush()*, *tcgetattr()*, *tcsetattr()*, *tcsetattr()*, *ttyname()*

11818 POSIX_DEVICE_SPECIFIC_R: Thread-Safe General Terminal

11819 *ttyname_r()*

11820 POSIX_FD_MGMT: File Descriptor Management

11821 *dup()*, *dup2()*, *fcntl()*, *fgetpos()*, *fseek()*, *fseeko()*, *fsetpos()*, *ftell()*, *ftello()*, *ftruncate()*, *lseek()*,
 11822 *rewind()*

11823 POSIX_FIFO: FIFO

11824 *mkfifo()*

11825 POSIX_FILE_ATTRIBUTES: File Attributes

11826 *chmod()*, *chown()*, *fchmod()*, *fchown()*, *umask()*

11827 POSIX_FILE_LOCKING: Thread-Safe Stdio Locking

11828 *flockfile()*, *ftrylockfile()*, *funlockfile()*, *getc_unlocked()*, *getchar_unlocked()*, *putc_unlocked()*,

11829 *putchar_unlocked()*

11830 POSIX_FILE_SYSTEM: File System

11831 *access(), chdir(), closedir(), creat(), fpathconf(), fstat(), getcwd(), link(), mkdir(), opendir(),*

11832 *pathconf(), readdir(), remove(), rename(), rewinddir(), rmdir(), stat(), tmpfile(), tmpnam(),*

11833 *unlink(), utime()*

11834 POSIX_FILE_SYSTEM_EXT: File System Extensions

11835 *glob(), globfree()*

11836 POSIX_FILE_SYSTEM_R: Thread-Safe File System

11837 *readdir_r()*

11838 POSIX_JOB_CONTROL: Job Control

11839 *setpgid(), tcgetpgrp(), tcsetpgrp()*

11840 POSIX_MULTI_PROCESS: Multiple Processes

11841 *_Exit(), _exit(), assert(), atexit(), clock(), execl(), execle(), execlp(), execv(), execve(), execvp(),*

11842 *exit(), fork(), getpgrp(), getpid(), getppid(), setsid(), sleep(), times(), wait(), waitpid()*

11843 POSIX_NETWORKING: Networking

11844 *accept(), bind(), connect(), endhostent(), endnetent(), endprotoent(), endservent(),*

11845 *freeaddrinfo(), gai_strerror(), getaddrinfo(), gethostbyaddr(), gethostbyname(), gethostent(),*

11846 *gethostname(), getnameinfo(), getnetbyaddr(), getnetbyname(), getnetent(), getpeername(),*

11847 *getprotobyname(), getprotobynumber(), getprotoent(), getservbyname(), getservbyport(),*

11848 *getservent(), getsockname(), getsockopt(), h_errno, htonl(), htons(), if_freenameindex(),*

11849 *if_indextoname(), if_nameindex(), if_nametoindex(), inet_addr(), inet_ntoa(), inet_ntop(),*

11850 *inet_pton(), listen(), ntohl(), ntohs(), recv(), recvfrom(), recvmsg(), send(), sendmsg(), sendto(),*

11851 *sethostent(), setnetent(), setprotoent(), setservent(), setsockopt(), shutdown(), socket(),*

11852 *socketatmark(), socketpair()*

11853 POSIX_PIPE: Pipe

11854 *pipe()*

11855 POSIX_REGEX: Regular Expressions

11856 *regcomp(), regerror(), regexexec(), regfree()*

11857 POSIX_SHELL_FUNC: Shell and Utilities

11858 *pclose(), popen(), system(), wordexp(), wordfree()*

11859 POSIX_SIGNALS: Signal

11860 *abort(), alarm(), kill(), pause(), raise(), sigaction(), sigaddset(), sigdelset(), sigemptyset(),*

11861 *sigfillset(), sigismember(), signal(), sigpending(), sigprocmask(), sigsuspend(), sigwait()*

11862 POSIX_SIGNAL_JUMP: Signal Jump Functions

11863 *siglongjmp(), sigsetjmp()*

11864 POSIX_SINGLE_PROCESS: Single Process

11865 *confstr(), environ, errno, getenv(), setenv(), sysconf(), uname(), unsetenv()*

11866 POSIX_SYMBOLIC_LINKS: Symbolic Links

11867 *lstat(), readlink(), symlink()*

11868 POSIX_SYSTEM_DATABASE: System Database

11869 *getgrgid(), getgrnam(), getpwnam(), getpwuid()*

11870 POSIX_SYSTEM_DATABASE_R: Thread-Safe System Database

11871 *getgrgid_r(), getgrnam_r(), getpwnam_r(), getpwuid_r()*

11872	POSIX_USER_GROUPS: User and Group
11873	<i>getegid()</i> , <i>geteuid()</i> , <i>getgid()</i> , <i>getgroups()</i> , <i>getlogin()</i> , <i>getuid()</i> , <i>setegid()</i> , <i>seteuid()</i> , <i>setgid()</i> ,
11874	<i>setuid()</i>
11875	POSIX_USER_GROUPS_R: Thread-Safe User and Group
11876	<i>getlogin_r()</i>
11877	POSIX_WIDE_CHAR_DEVICE_IO: Device Input and Output
11878	<i>fgetwc()</i> , <i>fgetws()</i> , <i>fputwc()</i> , <i>fputws()</i> , <i>fwide()</i> , <i>fwprintf()</i> , <i>fwscanf()</i> , <i>getwc()</i> , <i>getwchar()</i> ,
11879	<i>putwc()</i> , <i>putwchar()</i> , <i>ungetwc()</i> , <i>vfwprintf()</i> , <i>vfwscanf()</i> , <i>vwprintf()</i> , <i>vwscanf()</i> , <i>wprintf()</i> ,
11880	<i>wscanf()</i>
11881	XSI_C_LANG_SUPPORT: XSI General C Library
11882	<i>_tolower()</i> , <i>_toupper()</i> , <i>a64l()</i> , <i>daylight()</i> , <i>drand48()</i> , <i>erand48()</i> , <i>ffs()</i> , <i>getcontext()</i> , <i>getdate()</i> ,
11883	<i>getsubopt()</i> , <i>hcreate()</i> , <i>hdestroy()</i> , <i>hsearch()</i> , <i>iconv()</i> , <i>iconv_close()</i> , <i>iconv_open()</i> , <i>initstate()</i> ,
11884	<i>insque()</i> , <i>isascii()</i> , <i>jrand48()</i> , <i>l64a()</i> , <i>lcong48()</i> , <i>lfind()</i> , <i>lrand48()</i> , <i>lsearch()</i> , <i>makecontext()</i> ,
11885	<i>memccpy()</i> , <i>mrand48()</i> , <i>nrand48()</i> , <i>random()</i> , <i>remque()</i> , <i>seed48()</i> , <i>setcontext()</i> , <i>setstate()</i> ,
11886	<i>signgam</i> , <i>srand48()</i> , <i>srandom()</i> , <i>strcasecmp()</i> , <i>strdup()</i> , <i>strfmon()</i> , <i>strncasecmp()</i> , <i>strptime()</i> ,
11887	<i>swab()</i> , <i>swapcontext()</i> , <i>tdelete()</i> , <i>tfind()</i> , <i>timezone()</i> , <i>toascii()</i> , <i>tsearch()</i> , <i>twalk()</i>
11888	XSI_DBM: XSI Database Management
11889	<i>dbm_clearerr()</i> , <i>dbm_close()</i> , <i>dbm_delete()</i> , <i>dbm_error()</i> , <i>dbm_fetch()</i> , <i>dbm_firstkey()</i> ,
11890	<i>dbm_nextkey()</i> , <i>dbm_open()</i> , <i>dbm_store()</i>
11891	XSI_DEVICE_IO: XSI Device Input and Output
11892	<i>fmsg()</i> , <i>poll()</i> , <i>pread()</i> , <i>pwrite()</i> , <i>readv()</i> , <i>writv()</i>
11893	XSI_DEVICE_SPECIFIC: XSI General Terminal
11894	<i>grantpt()</i> , <i>posix_openpt()</i> , <i>ptsname()</i> , <i>unlockpt()</i>
11895	XSI_DYNAMIC_LINKING: XSI Dynamic Linking
11896	<i>dlclose()</i> , <i>dlerror()</i> , <i>dlopen()</i> , <i>dlsym()</i>
11897	XSI_FD_MGMT: XSI File Descriptor Management
11898	<i>truncate()</i>
11899	XSI_FILE_SYSTEM: XSI File System
11900	<i>basename()</i> , <i>dirname()</i> , <i>fchdir()</i> , <i>fstatvfs()</i> , <i>ftw()</i> , <i>lchown()</i> , <i>lockf()</i> , <i>mknod()</i> , <i>mkstemp()</i> , <i>nftw()</i> ,
11901	<i>realpath()</i> , <i>seekdir()</i> , <i>statvfs()</i> , <i>sync()</i> , <i>telldir()</i> , <i>tempnam()</i>
11902	XSI_I18N: XSI Internationalization
11903	<i>catclose()</i> , <i>catgets()</i> , <i>catopen()</i> , <i>nl_langinfo()</i>
11904	XSI_IPC: XSI Interprocess Communication
11905	<i>ftok()</i> , <i>msgctl()</i> , <i>msgget()</i> , <i>msgrcv()</i> , <i>msgsnd()</i> , <i>semctl()</i> , <i>semget()</i> , <i>semop()</i> , <i>shmat()</i> , <i>shmctl()</i> ,
11906	<i>shmdt()</i> , <i>shmget()</i>
11907	XSI_JOB_CONTROL: XSI Job Control
11908	<i>tcgetsid()</i>
11909	XSI_JUMP: XSI Jump Functions
11910	<i>_longjmp()</i> , <i>_setjmp()</i>
11911	XSI_MATH: XSI Maths Library
11912	<i>j0()</i> , <i>j1()</i> , <i>jn()</i> , <i>scalb()</i> , <i>y0()</i> , <i>y1()</i> , <i>yn()</i>
11913	XSI_MULTI_PROCESS: XSI Multiple Process
11914	<i>getpgid()</i> , <i>getpriority()</i> , <i>getrlimit()</i> , <i>getrusage()</i> , <i>getsid()</i> , <i>nice()</i> , <i>setpgid()</i> , <i>setpriority()</i> ,
11915	<i>setrlimit()</i> , <i>ulimit()</i> , <i>usleep()</i> , <i>vfork()</i> , <i>waitid()</i>

11916	XSI_SIGNALS: XSI Signal
11917	<i>bsd_signal()</i> , <i>killpg()</i> , <i>sigaltstack()</i> , <i>sighold()</i> , <i>sigignore()</i> , <i>siginterrupt()</i> , <i>sigpause()</i> , <i>sigrelse()</i> ,
11918	<i>sigset()</i> , <i>ualarm()</i>
11919	XSI_SINGLE_PROCESS: XSI Single Process
11920	<i>gethostid()</i> , <i>gettimeofday()</i> , <i>putenv()</i>
11921	XSI_SYSTEM_DATABASE: XSI System Database
11922	<i>endpwent()</i> , <i>getpwent()</i> , <i>setpwent()</i>
11923	XSI_SYSTEM_LOGGING: XSI System Logging
11924	<i>closelog()</i> , <i>openlog()</i> , <i>setlogmask()</i> , <i>syslog()</i>
11925	XSI_THREAD_MUTEX_EXT: XSI Thread Mutex Extensions
11926	<i>pthread_mutexattr_gettype()</i> , <i>pthread_mutexattr_settype()</i>
11927	XSI_THREADS_EXT: XSI Threads Extensions
11928	<i>pthread_attr_getguardsize()</i> , <i>pthread_attr_getstack()</i> , <i>pthread_attr_setguardsize()</i> ,
11929	<i>pthread_attr_setstack()</i> , <i>pthread_getconcurrency()</i> , <i>pthread_setconcurrency()</i>
11930	XSI_TIMERS: XSI Timers
11931	<i>getitimer()</i> , <i>setitimer()</i>
11932	XSI_USER_GROUPS: XSI User and Group
11933	<i>endgrent()</i> , <i>endutxent()</i> , <i>getgrent()</i> , <i>getutxent()</i> , <i>getutxid()</i> , <i>getutxline()</i> , <i>pututxline()</i> ,
11934	<i>setgrent()</i> , <i>setregid()</i> , <i>setreuid()</i> , <i>setutxent()</i>
11935	XSI_WIDE_CHAR: XSI Wide-Character Library
11936	<i>wcswidth()</i> , <i>wcwidth()</i>

Index

/dev/tty.....	21	_POSIX_THREAD_PRIO_INHERIT.....	284
/etc/passwd	34	_POSIX_THREAD_PRIO_PROTECT	284
<pthread.h>	161	_POSIX_THREAD_PROCESS_SHARED.....	284
_asm_builtin_atoi().....	84	_POSIX_THREAD_SAFE_FUNCTIONS.....	284
_exit()	101, 119	_POSIX_THREAD_SPORADIC_SERVER	284
_Exit()	271	_POSIX_TIMEOUTS	284
_exit().....	271	_POSIX_TIMERS	284
_longjmp()	274	_POSIX_TRACE.....	284
_POSIX_ADVISORY_INFO	280	_POSIX_TRACE_EVENT_FILTER.....	285
_POSIX_ASYNCHRONOUS_IO	280	_POSIX_TRACE_INHERIT.....	285
_POSIX_BARRIERS.....	280	_POSIX_TRACE_LOG.....	285
_POSIX_CHOWN_RESTRICTED	4, 280	_POSIX_TYPED_MEMORY_OBJECTS	285
_POSIX_CLOCK_SELECTION	280	_POSIX_TZNAME_MAX	57
_POSIX_CPUTIME.....	281	_POSIX_VDISABLE	5
_POSIX_C_SOURCE.....	85, 89	_SC_PAGESIZE.....	118-119
_POSIX_FSYNC	281	_setjmp()	274
_POSIX_IPV6.....	281	_XOPEN_SOURCE	85
_POSIX_JOB_CONTROL	5, 281, 285	__errno().....	92
_POSIX_MAPPED_FILES.....	281	abort()	271
_POSIX_MEMLOCK.....	281	access.....	272
_POSIX_MEMLOCK_RANGE	281	access()	33
_POSIX_MEMORY_PROTECTION	282	active trace stream.....	203
_POSIX_MESSAGE_PASSING.....	282	adb, rationale for omission	261
_POSIX_MONOTONIC_CLOCK.....	282	address families	177
_POSIX_NO_TRUNC	5	addressing.....	177
_POSIX_PRIORITIZED_IO	282	advisory information	104
_POSIX_PRIORITY_SCHEDULING.....	282	aio_cancel().....	113-114
_POSIX_REALTIME_SIGNALS	282	aio_fsync().....	97, 112
_POSIX_REGEX	282	AIO_LISTIO_MAX.....	285
_POSIX_RTSIG_MAX	94, 287	AIO_MAX	285
_POSIX_SAVED_IDS	5, 282	AIO_PRIO_DELTA_MAX.....	285
_POSIX_SEMAPHORES	282	aio_read().....	114
_POSIX_SHARED_MEMORY_OBJECTS	282	aio_suspend()	112, 138
_POSIX_SHELL.....	283	aio_write()	114
_POSIX_SOURCE.....	85	alarm	271
_POSIX_SPAWN	283	alarm().....	102, 137
_POSIX_SPINLOCKS	283	alias	230, 274
_POSIX_SPORADIC_SERVER	283	alias substitution.....	233
_POSIX_SS_REPL_MAX.....	133	AND lists.....	248
_POSIX_SYNCHRONIZED_IO	283	application instrumentation.....	191
_POSIX_SYNC_IO	281	appropriate privilege.....	13, 25
_POSIX_THREADS.....	283	ar	277-278
_POSIX_THREAD_ATTR_STACKADDR.....	283	arbitrary file size	228
_POSIX_THREAD_ATTR_STACKSIZE	283	ARG_MAX	23, 222, 285
_POSIX_THREAD_PRIORITY_SCHEDULING	284	arithmetic expansion	240
		as, rationale for omission	261

- asa 275, 277-278
- ASCII 24
- async-cancel safety 175
- async-signal-safe 101
- asynchronous error 178
- asynchronous I/O 112, 272
- asynchronous lists 248
- at 275
- atexit() 174
- atoi() 84
- awk 275, 277
- background 19-21, 67-68
- backslash 231
- banner, rationale for omission 261
- barriers 154
- basename 274-275
- basic regular expression 60
- batch 275
 - general concepts 258
- batch environment 255
 - option definitions 256
- batch environment utilities
 - common behavior 261
- batch services 260
- batch systems
 - historical implementations 255
 - history 255
- bc 274-275, 279
- BC_BASE_MAX 223
- BC_DIM_MAX 223
- BC_SCALE_MAX 223
- bg 230, 275
- bounded response 273
- bracket expression
 - grammar 64
- BRE
 - expression anchoring 62
 - grammar lexical conventions 64
 - matching a collating element 60
 - matching a single character 60
 - matching multiple characters 62
 - ordinary character 60
 - periods 60
 - precedence 62
 - special character 60
- BSD 16, 18, 21, 24, 67-68, 93-96, 100, 204
- built-in utilities 229
- C Shell 18-20
- C-language extensions 269, 274
- c99 277
- calendar, rationale for omission 261
- cancel, rationale for omission 261
- canonical mode input processing 68
- case 249
- case folding 34-35
- cat 228-229
- catclose() 276
- catgets() 276
- catopen() 276
- cd 230, 275
- CEO 61
- change history 77, 219
- character 14
- character encoding 42
- character set 41
- character set description file 42
- character set, portable filename 24
- character, rationale 14
- CHARCLASS_NAME_MAX 47
- CHAR_MAX 49
- chgrp 228, 275-276
- CHILD_MAX 5, 205, 223, 248, 285
- chmod 228, 272, 275-276
- chmod() 26
- chown 228, 272, 275-276
- chown() 26
- chroot() 25
- chroot, rationale for omission 261
- cksum 228-229, 275-276
- clock 134
- clock tick 14
- clock tick, rationale 14
- CLOCKRES_MIN 285
- clocks 134
- clock_getcpuclockid() 140, 142
- clock_nanosleep() 138
- CLOCK_PROCESS_CPUTIME_ID 140, 142
- CLOCK_REALTIME 134-138
- CLOCK_THREAD_CPUTIME_ID 140, 142
- close 272
- close() 118-119
- closedir 272
- closelog() 276
- cmp 228-229, 275
- codes 8, 83, 221
- col, rationale for omission 261
- collating element order 61
- COLL_WEIGHTS_MAX 223
- column position 15
- COLUMNS 56
- comm 275
- command 14, 230, 274

command execution.....	246	direct I/O	16
command language	269, 274	directory	16
command search.....	246	directory device	65
command substitution	239	directory entry.....	16
compilation environment	85	directory files.....	65
complex data manipulation.....	269, 275	directory protection	33
compound commands.....	249	directory structure.....	65
concurrent execution	33	directory, root.....	25
conditional construct		dirname.....	274-275
case	249	dis, rationale for omission	262
if	250	display.....	16
configurable limits	279, 285	dot	16
configuration interrogation.....	268, 271	dot-dot	16, 24, 38
configuration options	277	double-quotes.....	231
shell and utilities	278	du	229, 275-276
system interfaces.....	280	dup.....	272
conformance	5, 9, 12-13, 35, 77, 203, 219	dup()	119
conformance document.....	5	dup2.....	272
conformance document, rationale	5	dup2().....	119
conforming application	12, 94, 226, 228	EBUSY.....	92, 161
conforming application, strictly.....	9, 12, 100	ECANCELED	90
confstr	272	echo.....	274
connection indication queue	178	ECHOE	71
control mode.....	70	ECHOK.....	71
controlling terminal.....	15, 67, 271	ECHONL	70
core	34	ed	275-276
covert channel	35	EDOM	92
cp	228, 275	EFAULT	90
cpio, rationale for omission	261	effective user ID	33
cpp, rationale for omission.....	262	EFTYPE.....	90
creat()	118, 228	EILSEQ.....	92
crontab	275	EINPROGRESS	112
CSIZE	70	EINTR.....	90, 93, 102
csplit	275	EINVAL	90
ctags.....	277	ELOOP	91
ctime()	273	emacs, rationale for omission	262
cu, rationale for omission	262	ENAMETOOLONG	91
cut	275	endgrent()	32
data access	268, 272	endpwent()	32
data type.....	203	ENOMEM.....	91
date	276	ENOSYS	91, 115
dc, rationale for omission	262	ENOTSUP	91
dd	228-229, 275	ENOTTY	65, 90-91
definitions	83, 220	env	274, 276
DELAYTIMER_MAX	285	environment access.....	268, 272
determinism.....	268	environment variable.....	55
device number.....	15	definition	55
device, logical.....	22	EOVERFLOW	91
df	229, 275-276	EPERM.....	170
diff.....	275	EPIPE.....	92
dircmp, rationale for omission	262	Epoch	17, 39, 134

ERANGE.....	92	FIFO	17, 24, 128
ERASE.....	68	FIFO special file.....	17
ERE	63	file.....	17
alternation	63	file access permissions.....	33
bracket expression.....	63	file classes	17
expression anchoring.....	64	file descriptors.....	103
grammar.....	64	file format notation.....	41
grammar lexical conventions.....	64	file hierarchy	33
matching a collating element.....	63	file hierarchy manipulation.....	269, 275
matching a single character.....	63	file permissions	33, 67
matching multiple characters	63	file removal	221
ordinary character	63	file size, arbitrary	228
periods	63	file system.....	17
precedence	64	file system, mounted.....	22
special character	63	file system, root	25
EROFS	92	file times update	35
errno	89	file, passwd	24
per-thread.....	92	filename	17, 34
error conditions.....	40	fileno()	23
error handling.....	253	filtering of trace event types.....	200
error numbers.....	89, 93	find	275-276
errors	244	FIPS requirements	4
escape character.....	231	flockfile().....	93
ex	274-276	fnmatch()	277
exec.....	197	fold.....	275
exec family	19, 116, 174, 224, 229, 251, 271	fopen.....	272
Execution Time Monitoring	139	fopen()	18, 228
execution time, measurement.....	35	for loop.....	249
exit status	244	foreground.....	19-21, 66-68
exit().....	101, 271	fork()	19, 67, 108, 116, 118-119, 197, 204, 206, 271
EXIT_FAILURE	273	fort77	277
EXIT_SUCCESS	273	fpathconf	272
expand	275	fpathconf().....	271
expr.....	274-275	fputc	272
EXPR_NEST_MAX.....	223	fread.....	272
extended regular expression	63	free()	101, 173
extended security controls.....	33	fseek.....	272
false	230, 274	fseeko().....	204
fc	230, 274	fsetpos.....	272
fchmod.....	272	fsetpos()	204
fclose.....	272	fstat	271-272
fcntl.....	272	fsync().....	112
fcntl()	66, 90, 119, 121, 204	ftello().....	204
fcntl() locks.....	176	ftruncate	272
fdopen().....	119	ftruncate().....	118, 120
FD_CLOEXEC	121	ftw()	228
feature test macro	85-86, 204	function definition command	250
fg	230, 275	functions	
fgetc	272	implementation of.....	84
fgetpos().....	204	use of.....	84
field splitting.....	241	fwrite	272

gencat	276	iconv()	276
general terminal interface	65	iconv_close()	276
getc	272	iconv_open()	276
getc()	164	id	276
getch	272	if	250
getconf	222, 271, 276-277	implementation	18
getegid	271	implementation, historical	18
geteuid	271	implementation, hosted	18
getgid	271	implementation, native	23
getgrent()	32	implementation, specific	18
getgrgid	271	implementation-defined	5-7
getgrgid()	32, 165	implementation-defined, rationale	5
getgrnam	271	incomplete pathname	18
getgrnam()	32, 35, 165	input file descriptor	
getgroups()	26	duplication	243
getlogin	271	input mode	70
getopt()	73, 276-277	input processing	68
getopts	230, 277	canonical mode	68
getpgrp()	20	non-canonical mode	69
getpid	271	inter-user communication	270, 276
getpid()	102	interactive facilities	269, 274
getppid	271	interface characteristics	66
getpriority()	129	interfaces	177
getpwent()	32	internationalization variable	55
getpwnam	271	interprocess communication	103
getpwnam()	32, 35, 165	invalid	
getpwuid	271	use in RE	59
getpwuid()	32, 165	ioctl()	65, 91
getrlimit()	229	IPC	103
getrusage()	141	ISO C standard	12, 14, 39, 66, 84-86
getty	67		88-89, 92, 94, 101, 204
getuid	271	ISO / IEC 646: 1991 standard	24
getuid()	102, 205	ISTRIP	70
gid_t	32	job control	18-21, 23, 66-68, 94, 100, 271
glob()	277	job control, implementing applications	21
globfree()	277	job control, implementing shells	19
gmtime()	39	job control, implementing systems	21
grammar conventions	225	jobs	230, 275
grep	275-276	join	275
group database	17	kernel	22
group database access	32	kernel entity	167
group file	18	kill	230, 275
grouping commands	249	kill()	93-95, 98, 100-101, 204
head	275	last close	119
headers	74	lchown()	26
here-document	243	LC_COLLATE	47
historical implementations	18	LC_CTYPE	46
HOME	5	LC_MESSAGES	51
host byte order	35	LC_MONETARY	49
hosted implementation	18	LC_NUMERIC	50, 73
ICANON	68, 71	LC_TIME	50

ld, rationale for omission	262	mapped	22
legacy	6	margin code notation	8
legacy, rationale	6	margin codes	8, 83, 221
lex	277-278	mathematical functions	
library routine	22	error conditions	40
limits	222	NaN arguments	40
line, rationale for omission	262	MAX_CANON	68, 223, 286
LINES	56	MAX_INPUT	223, 286
LINE_MAX	23, 31, 223	may	5
link	272	may, rationale	5
link()	16, 26	MCL_FUTURE	116
LINK_MAX	223, 286	MCL_INHERIT	117
lint, rationale for omission	262	memory locking	114
lio_listio()	97, 113	memory management	114, 272
lists	247	memory management unit	115
ln	228, 275	memory object	22
local mode	70	memory synchronization	36
locale	44, 276	memory-resident	22
definition example	52	mesg	276
definition grammar	51	message passing	106, 272-273
grammar	51	message queue	106
lexical conventions	51	mkdir	272, 275
locale configuration	269, 276	mkfifo	275
locale definition	45	mkfifo()	18
localedef	276-277	mknod()	18
localtime()	39, 164	mknod, rationale for omission	263
logger	276	mktime()	39
logical device	22	mlockall()	116
login, rationale for omission	262	mmap()	118-122
LOGIN_NAME_MAX	286	MMU	115
LOGNAME	5, 57	modem disconnect	69
logname	276	monotonic clock	138
longjmp()	91, 101, 171, 173, 274	more	274, 276
LONG_MAX	72	mount point	22
LONG_MIN	72	mount()	22
lorder, rationale for omission	262	mounted file system	22
lp	276	mprotect()	118
lpstat, rationale for omission	262	mq_open()	107
ls	229, 275	MQ_OPEN_MAX	286
lseek	272	MQ_PRIO_MAX	286
lseek()	112-113, 119, 163, 204	mq_receive()	107
lstat	271-272	mq_send()	107
lstat()	26, 228	mq_timedreceive()	139
lutime()	26	mq_timedsend()	139
macro	7	msg*()	103
mail, rationale for omission	262	msgctl()	104
mailx	274, 276	msgget()	104
make	277-278	msgrcv()	104
malloc()	101, 122-123, 152, 164-165, 173, 175	msgsnd()	104
man	274	msync()	118
map	22	multi-byte character	17, 68, 70

multiple tasks.....	269, 275
munmap().....	118, 120, 122, 125
mutex	156
mutex attributes	
extended.....	159
mutex initialization.....	170
mv.....	228, 275
name space.....	86
name space pollution	85-86
NAME_MAX.....	223, 286
NaN arguments	40
nanosleep().....	135, 137-138, 273
native implementation	23
network byte order.....	35
newgrp.....	276
news, rationale for omission	263
NGROUPS_MAX.....	4, 25, 223, 280, 286
nice.....	275
nice value.....	23
nice().....	129
nl_langinfo().....	273
nm.....	277
noclobber option.....	242
nohup	275
non-canonical mode input processing	69
non-printable	31
normative references.....	5, 77, 219
NQS	256
obsolescent.....	6
obsolescent, rationale.....	6
od	275, 277
open.....	272
open file description.....	23
open file descriptors.....	244
open().....	18, 67, 118-121, 228
opendir.....	272
openlog().....	276
OPEN_MAX	5, 223, 286-287
option definitions	256
optional behavior	288
options	178
OR lists.....	248
orphaned process group	23, 100
output device.....	65
output file descriptor	
duplication.....	243
output mode	70
output processing.....	69
overflow conditions	203
overflow in dumping trace streams	203
overflow in trace streams	203
pack, rationale for omission.....	263
page.....	24, 118, 121
PAGESIZE	118, 162, 286
parallel I/O	163
parameter expansion	238
parameters	70, 234
parameters, positional.....	234
parameters, special.....	234
parent directory	24
passwd file	24
passwd, rationale for omission.....	263
paste	275
patch	275-277
PATH	57
pathchk	275
pathconf.....	272
pathconf().....	18, 222, 271
pathname expansion.....	242
pathname resolution.....	37
pathname, incomplete	18
PATH_MAX	91, 223, 286
pattern matching	
multiple character	254
notation	253
single character	253
patterns	
filename expansion	254
pause	271
pause().....	99, 102
pax.....	275-277
pcat, rationale for omission	263
pclose().....	277
pending error.....	178
per-thread errno.....	92
performance enhancements	268
pg, rationale for omission	263
PID_MAX	204
pipe.....	19, 24, 272
pipe()	100
pipelines	247
PIPE_BUF.....	223, 286
popen().....	274, 277-278
portability	8, 83, 221
portability codes.....	8, 83, 221
portable character set.....	41
portable filename character set.....	24
positional parameters	234
POSIX locale	45
POSIX.1 symbols.....	85
POSIX.13.....	122
POSIX2_BC_BASE_MAX.....	279

POSIX2_BC_DIM_MAX.....	279	POSIX_MADV_WILLNEED	105
POSIX2_BC_SCALE_MAX.....	279	posix_mem_offset()	122-123
POSIX2_BC_STRING_MAX	279	POSIX_MULTI_PROCESS.....	293
POSIX2_CHAR_TERM	279	POSIX_NETWORKING	293
POSIX2_COLL_WEIGHTS_MAX.....	279	POSIX_PIPE.....	293
POSIX2_C_BIND.....	224, 278	POSIX_REC_INCR_XFER_SIZE	105
POSIX2_C_DEV.....	224, 278	POSIX_REC_MAX_XFER_SIZE.....	105
POSIX2_EXPR_NEST_MAX	279	POSIX_REC_MIN_XFER_SIZE	105
POSIX2_FORT_DEV.....	224, 278	POSIX_REC_XFER_ALIGN	105
POSIX2_FORT_RUN	224, 278	POSIX_REGEXP.....	293
POSIX2_LINE_MAX.....	279	POSIX_SHELL_FUNC.....	293
POSIX2_LOCALEDEF	224, 276, 278	POSIX_SIGNALS.....	293
POSIX2_PBS.....	279	POSIX_SIGNAL_JUMP.....	293
POSIX2_PBS_ACCOUNTING.....	279	POSIX_SINGLE_PROCESS.....	293
POSIX2_PBS_CHECKPOINT	279	posix_spawn().....	206, 271
POSIX2_PBS_LOCATE.....	279	posix_spawnp()	206, 271
POSIX2_PBS_MESSAGE.....	279	POSIX_SYMBOLIC_LINKS.....	293
POSIX2_PBS_TRACK.....	279	POSIX_SYSTEM_DATABASE.....	293
POSIX2_RE_DUP_MAX	280	POSIX_SYSTEM_DATABASE_R.....	293
POSIX2_SW_DEV	224, 278	posix_trace_eventid_open()	198
POSIX2_SYMLINKS	223	POSIX_TRACE_LOOP	203
POSIX2_UPE.....	224, 278-279	posix_typed_mem_get_info()	122
POSIX2_VERSION	280	posix_typed_mem_open()	122
POSIX_ALLOC_SIZE_MIN	105	POSIX_USER_GROUPS.....	294
POSIX_C_LANG_JUMP	291	POSIX_USER_GROUPS_R.....	294
POSIX_C_LANG_MATH	291	POSIX_VERSION	286
POSIX_C_LANG_SUPPORT	292	POSIX_WIDE_CHAR_DEVICE_IO	294
POSIX_C_LANG_SUPPORT_R.....	292	post-mortem filtering of trace event types.....	200
POSIX_C_LANG_WIDE_CHAR	292	pr	275-276
POSIX_C_LIB_EXT	292	pread()	163
POSIX_DEVICE_IO	292	printf.....	274-275
POSIX_DEVICE_SPECIFIC.....	292	printing.....	270
POSIX_DEVICE_SPECIFIC_R.....	292	privilege.....	33
posix_fadvise()	105	process group	66
POSIX_FADV_DONTNEED.....	105	process group ID.....	19, 66-67
POSIX_FADV_NOREUSE.....	105	process group lifetime	67
POSIX_FADV_RANDOM	105	process group, orphaned.....	23, 100
POSIX_FADV_SEQUENTIAL	105	process groups, concepts in job control.....	19
POSIX_FADV_WILLNEED.....	105	process ID reuse.....	39
POSIX_FD_MGMT.....	292	process ID, rationale	204
POSIX_FIFO.....	292	process management.....	268, 271
POSIX_FILE_ATTRIBUTES.....	292	process scheduling.....	127, 271
POSIX_FILE_LOCKING	292	prof, rationale for omission	263
POSIX_FILE_SYSTEM.....	293	profiling	277
POSIX_FILE_SYSTEM_EXT.....	293	programming manipulation	193
POSIX_FILE_SYSTEM_R.....	293	prompting	236
POSIX_JOB_CONTROL.....	293	protocols.....	177
posix_madvise().....	105	ps	275-276
POSIX_MADV_DONTNEED.....	105	pthread.....	201
POSIX_MADV_RANDOM.....	105	pthread_attr_getguardsize().....	163
POSIX_MADV_SEQUENTIAL.....	105	pthread_attr_setguardsize()	163

PTHREAD_BARRIER_SERIAL_THREAD	154	pwd.....	275
pthread_barrier_wait()	155, 174	pwrite()	163
pthread_cond_init()	151	queuing of waiting threads	175
pthread_cond_timedwait()	93, 138, 160, 280	quote removal	242
pthread_cond_wait()	93, 109, 160	quoting	231
pthread_create()	151-152	rand()	173
PTHREAD_CREATE_DETACHED	172	RCS, rationale for omission	263
PTHREAD_DESTRUCTOR_ITERATIONS	287	RE	
pthread_detach()	172	grammar	64
pthread_getconcurrency()	162	RE bracket expression	60
pthread_getcpuclockid()	140, 142	read	230, 272, 274
pthread_join()	93, 172	read lock	161
PTHREAD_KEYS_MAX	287	read()	19, 67-68, 90, 99-100, 102
pthread_key_create()	153		112-114, 117, 119, 163, 173, 205
pthread_mutexattr_gettype()	160	read-write attributes	160
pthread_mutexattr_settype()	160	read-write locks	160
PTHREAD_MUTEX_DEFAULT	159	readdir	272
PTHREAD_MUTEX_ERRORCHECK	159	reading an active trace stream	203
pthread_mutex_init()	151	reading data	68
pthread_mutex_lock()	93, 159, 173	readlink	272
PTHREAD_MUTEX_NORMAL	159	readlink()	26
PTHREAD_MUTEX_RECURSIVE	159	realpath	272
pthread_mutex_timedlock()	139	realtime	104
pthread_mutex_trylock()	159	realtime signal delivery	96
pthread_mutex_unlock()	159	realtime signal generation	96
PTHREAD_PROCESS_PRIVATE	161	realtime signals	110
PTHREAD_PROCESS_SHARED	161	red, rationale for omission	263
pthread_rwlockattr_destroy()	161	redirect input	243
pthread_rwlockattr_getpshared()	161	redirect output	243
pthread_rwlockattr_init()	161	redirection	242
pthread_rwlockattr_setpshared()	161	references	5, 77, 219
pthread_rwlock_init()	161	regcomp()	277
PTHREAD_RWLOCK_INITIALIZER	161	regerror()	277
pthread_rwlock_rdlock()	161	regexec()	277
pthread_rwlock_t	161	regfree()	277
pthread_rwlock_tryrdlock()	161	regular expression	58
pthread_rwlock_trywrlock()	161	definitions	58
pthread_rwlock_unlock()	162, 175	general requirements	59
pthread_rwlock_wrlock()	161	grammar	64
pthread_self()	153	regular file	25
pthread_setconcurrency()	162	rejected utilities	261
pthread_setprio()	170	remove()	26
pthread_setschedparam()	170	rename	272
pthread_setspecific()	153	rename()	26
pthread_spin_lock()	155, 174	renice	275
pthread_spin_trylock()	155	replenishment period	131
PTHREAD_STACK_MIN	287	reserved words	234
PTHREAD_THREADS_MAX	287	rewinddir	272
putc	272	RE_DUP_MAX	223
putc()	164	rm	228, 275
putchar	272	rmdir	272, 275

<code>rmdir()</code>	26, 92	<code>sh</code>	276, 283
root directory.....	25, 38	<code>shall</code>	6
root file system.....	25	<code>shall, rationale</code>	6
root of a file system.....	25	<code>shar, rationale for omission</code>	263
routing.....	177	<code>shared memory</code>	119
<code>rsh, rationale for omission</code>	263	<code>SHELL</code>	57
<code>RTSIG_MAX</code>	287	<code>shell</code>	18-21, 66-68, 94, 100
<code>samefile()</code>	204	<code>shell commands</code>	244
<code>SA_NOCLDSTOP</code>	20	<code>shell errors</code>	244
<code>SA_SIGINFO</code>	98	<code>shell execution environment</code>	234, 253
scheduling allocation domain.....	168	<code>shell grammar</code>	251
scheduling contention scope.....	168-169	lexical conventions.....	252
scheduling documentation.....	169	rules.....	252
scheduling policy.....	39	<code>shell variables</code>	235
<code>SCHED_FIFO</code>	129-130, 168, 175, 273	<code>shell, job control</code>	19, 94, 100
<code>SCHED_OTHER</code>	129	<code>shl, rationale for omission</code>	263
<code>SCHED_RR</code>	129, 168, 175, 273	<code>shm*()</code>	103
<code>SCHED_SPORADIC</code>	273	<code>shmctl()</code>	104
scope.....	3, 77, 219	<code>shmdt()</code>	104
<code>sdb, rationale for omission</code>	263	<code>shm_open()</code>	119-121
<code>sdiff, rationale for omission</code>	263	<code>shm_unlink()</code>	120-121
seconds since the Epoch.....	39	<code>should</code>	6
security considerations.....	13, 17, 21, 32-33, 67	<code>should, rationale</code>	6
security, monolithic privileges.....	13	<code>SIGABRT</code>	30, 93
<code>sed</code>	275-276	<code>sigaction()</code>	96, 98
<code>sem*()</code>	103	<code>SIGBUS</code>	30, 93
semaphore.....	40, 108, 273	<code>SIGCHLD</code>	20, 95, 99-100
<code>semctl()</code>	104	<code>SIGCLD</code>	99-100
<code>semget()</code>	104	<code>SIGCONT</code>	20, 96, 99-100
<code>semop()</code>	104	<code>SIGEMT</code>	93
<code>sem_init()</code>	108	<code>SIGEV_NONE</code>	96
<code>SEM_NSEMS_MAX</code>	287	<code>SIGEV_SIGNAL</code>	96-97
<code>sem_open()</code>	108	<code>SIGFPE</code>	30, 94, 96, 98
<code>sem_timedwait()</code>	139	<code>SIGHUP</code>	100
<code>sem_trywait()</code>	93, 110	<code>SIGILL</code>	30, 94
<code>SEM_VALUE_MAX</code>	287	<code>SIGINT</code>	21, 166
<code>sem_wait()</code>	93, 110	<code>SIGIOT</code>	93
sequential lists.....	248	<code>SIGKILL</code>	93, 96, 100
session.....	20, 24, 67	<code>siglongjmp()</code>	91, 101, 274
<code>set</code>	236	<code>signal</code>	25
<code>setgid()</code>	25	<code>signal acceptance</code>	95
<code>setgrent()</code>	32	<code>signal actions</code>	99
<code>setjmp()</code>	274	<code>signal concepts</code>	93
<code>setlocale()</code>	273	<code>signal delivery</code>	95
<code>setlogmask()</code>	276	<code>signal generation</code>	95
<code>setpgid()</code>	19-20, 66-67	<code>signal names</code>	93
<code>setpriority()</code>	129	<code>signal()</code>	93, 95
<code>setpwent()</code>	32	<code>signals</code>	178, 253
<code>setrlimit()</code>	229	<code>SIGPIPE</code>	30, 92
<code>setsid()</code>	66	<code>sigprocmask()</code>	95
<code>setuid()</code>	25	<code>SIGRTMAX</code>	97, 99

SIGRTMIN	97, 99	SS_REPL_MAX	133
SIGSEGV	30, 94, 98	standard I/O streams	103
sigsetjmp()	274	stat	228, 271-272
sigset_t	93	stat()	15, 118, 228
SIGSTOP	100	state-dependent character encoding	42
sigsuspend()	99, 102	statvfs()	229
SIGSYS	93	STREAMS	103
SIGTERM	93	STREAM_MAX	287
sigtimedwait()	93, 112, 138	strings	277
SIGTRAP	93	strip	277
SIGTSTP	21, 100	structures, additions to	86
SIGTTIN	20, 68, 100	stty	56, 276
SIGTTOU	20, 68, 100	su, rationale for omission	264
SIGUSR1	93	subprofiling	11
SIGUSR2	93	subprofiling option groups	291
sigwait()	93, 173	subshells	20
sigwaitinfo()	93, 112	successfully completed	31
sigwait_multiple()	95	sum	228-229
SIG_DFL	95-96, 99	sum, rationale for omission	264
SIG_IGN	20, 95-96, 99, 102	superuser	13, 25, 33, 236, 261
simple commands	245	supplementary group ID	25
single-quotes	231	supplementary groups	33
size, rationale for omission	264	symbolic constant	7
SI_USER	98	symbolic link	26
sleep	271, 274	symbolic name	7
sleep()	101-102, 273	symbols	85
socket I/O mode	177	SYMLOOP_MAX	91
socket out-of-band data state	178	synchronized I/O	113, 272
socket owner	178	data integrity completion	31, 113
socket queue limit	178	file integrity completion	31, 113
socket receive queue	178	synchronously-generated signal	30
socket types	177	sysconf()	18, 115, 118-119, 166, 222, 271-272
sockets	177	syslog()	276
Internet Protocols	178	system call	30
IPv4	178	system documentation	6
IPv6	178	system environment	270, 276
local UNIX connection	178	System III	24, 204
software development	270, 277	system interfaces	206
sort	275-276	system reboot	31
spawn example	206	System V	16, 21, 94-95, 99-100
special built-in	250	system()	274, 277-278
special built-in utilities	255	tabs	275
special characters	69	tail	275
special control character	71	talk	274, 276
special parameters	234	tar, rationale for omission	264
specific implementation	18	tcgetattr()	20
spell, rationale for omission	264	tcgetpgrp()	20, 66-67
spin locks	155-156	tcsetattr()	20, 65
split	275	tcsetpgrp()	19-20
sporadic server scheduling policy	131	tee	274
SSIZE_MAX	205, 287	TERM	56

terminal access control	67	trace stream states	187
terminal device file	66	tracing	40, 179
closing	69	tracing all processes	187
terminal type	65	tracing, detailed objectives	180
terminology	5, 83, 220	triggering	202
termios structure	70	troff	275
test	274-275	trojan horse	13
TeX	275	true	230, 274
text file	31	tty	276
thread	31	ttyname()	164
thread cancelability states	173	TTY_NAME_MAX	287
thread cancelability type	173	typed memory	122
thread concurrency level	162	TZ	57
thread creation attributes	150	TZNAME_MAX	287
thread ID	32, 166	tzset()	273
thread interactions	177	ualarm	271
thread mutex	167	UID_MAX	205
thread read-write lock	175	uid_t	32
thread scheduling	167	ULONG_MAX	222
thread stack guard size	162	umask	230, 276
thread-safe	32	umask()	271
thread-safe function	32	umount()	22
thread-safety	40, 163	unalias	230, 274
thread-safety, rationale	40	uname	276
thread-specific data	152	uname()	271
threads	150	unbounded priority inversion	170
implementation models	152	undefined	6
tilde expansion	237	undefined, rationale	6
time	274, 277	unexpand	275
time()	90	uniq	275-276
timeouts	143	unlink	272
timers	134	unlink()	26, 92, 118, 120-121
TIMER_ABSTIME	135-136	unpack, rationale for omission	264
TIMER_MAX	287	unsafe functions	101
timer_settime()	135-136	unspecified	6
times()	90, 141, 271	unspecified, rationale	6
timestamp clock	202	until loop	250
time_t	39	user database	32
token recognition	233	user database access	32
TOSTOP	20	user requirements	267
touch	228, 275	usleep	271
tput	276	utility	40
tr.	275-276	utility argument syntax	71
trace analyzer	191	utility conventions	71
trace event type-filtering	200	utility description defaults	225
trace event types	200	utility limits	222
trace examples	189	utility syntax guidelines	72
trace model	184	utime	272
trace operation control	189	utime()	26
trace storage	188	uucp	276
trace stream attribute	194	uudecode	275-276

uuencode	275-276	XSI_THREADS_EXT.....	295
variable assignment	41	XSI_THREAD_MUTEX_EXT.....	295
variables	234	XSI_TIMERS	295
VEOF	71	XSI_USER_GROUPS.....	295
VEOL	71	XSI_WIDE_CHAR.....	295
Version 7	38, 231	yacc	277-278
vhangup()	21		
vi	274, 276		
virtual processor	32		
VMIN	71		
VTIME.....	71		
wait.....	230, 274		
wait().....	90, 94, 99, 101-102, 271		
waitpid()	20, 24, 101, 204, 271		
wall, rationale for omission	264		
wc.....	275		
WERASE.....	68		
while loop.....	250		
who	276		
wide-character codes	42		
word expansions.....	237		
wordexp()	277		
wordfree().....	277		
write.....	272, 274, 276		
write lock.....	161		
write().....	19-20, 67-68, 90, 99-100		
.....	102, 112-114, 117, 119, 205		
writing data.....	69		
WUNTRACED	20		
xargs	274		
XSI.....	33		
XSI IPC	103		
XSI supported functions	157		
XSI threads extensions	158		
XSI_C_LANG_SUPPORT	294		
XSI_DBM	294		
XSI_DEVICE_IO	294		
XSI_DEVICE_SPECIFIC.....	294		
XSI_DYNAMIC_LINKING	294		
XSI_FD_MGMT	294		
XSI_FILE_SYSTEM.....	294		
XSI_I18N.....	294		
XSI_IPC.....	294		
XSI_JOB_CONTROL	294		
XSI_JUMP.....	294		
XSI_MATH.....	294		
XSI_MULTI_PROCESS.....	294		
XSI_SIGNALS	295		
XSI_SINGLE_PROCESS.....	295		
XSI_SYSTEM_DATABASE	295		
XSI_SYSTEM_LOGGING	295		

